FPGA Operating System for Hard Real Time Applications

POLITECNICO DI TORINO

SCUOLA DI DOTTORATO
Dottorato in Meccatronica – XX ciclo

Tesi di Dottorato

# FPGA Operating System for Hard Real Time Applications

System Architecture and Application


**Gianluca Botto**

**Tutore**
Ing. Marcello Chiaberge

**Coordinatore del corso di dottorato**
Prof. Giancarlo Genta

Maggio 2008

# Contents

# Part I

# FPGA Operating System

# Chapter 1

# Context

## 1.1 State of the art

In mechatronics, as in many others fields, one of the main aspect is the prototyping. Since the mechatronics covers a lot of complex applications, the availability of a common digital platform to use in all of them is a valid help in the prototyping phase of the project.
On the market two companies are producing advanced prototyping systems:

- dSPACE with the `MicroAutoBox` [3] for automotive applications and the `RapidPro` [4] for general purpose applications and for `MicroAutoBox` expansion
- National Instruments with the `CompactRIO` [11]


Figure 1.1. dSPace MicroAutoBox picture taken from dSPACE web site

The dSPACE MicroAutoBox (fig.1.1)is a rapid prototyping module intended to perform fast function related to automotive applications, such as chassis control, powertrains, body control and X-by-wire applications.

MicroAutoBox is based on the `IBM PPC 750FX` processor which runs at a frequency of $800MHz$.
Algorithms can be programmed directly in Matlab/Simulink environment and downloaded by means of a custom host PC link. A dedicated block-set for typical automotive applications is provided but the user cannot modify this functions.
The dSPACE RapidPro (fig.1.2) are modules that can be used as expansion of the MicroAutoBox system. Three modules are available: the RapidPro Signal Conditioning unit, the RapidPro power stage unit and the RapidPro control unit.
The dSPACE RapidPro control unit can be used to add advanced I/O functional-

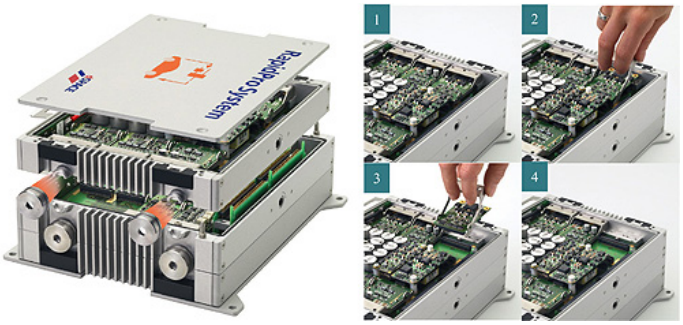



Figure 1.2. dSPACE RapidPro system picture taken from dSPACE web site

ities to the MicroAutoBox system.
The dSPACE RapidPro control unit is based on the `Freescale MPC565` processor which runs at a frequency of $40MHz$ (optionally $56MHz$).
The strength of these systems is the user programmability interface: it is very easy and based on the well known Simulink environment; the disadvantage using these systems is that the system architecture is closed: this means limited versatility.

The National Instruments CompactRIO (fig.1.3) is a general purpose prototyping system oriented, but not limited to, the industrial automation.
CompactRIO is based on a real time processor running at $400MHz$ and a reconfigurable FPGA (Field Programmable Gate Array) used for I/O interface and controls. The programmability is facilitated by the use of the graphical, proprietary, LabView environment. Again the architecture is closed and the user is strongly limited by the use of high level programming tools.
CompactRIO is, maybe, the prototyping system, that uses a FPGA, known by most people. The FPGA devices offers the maximum hardware flexibility at the moment; from now on the hardware configuration performed by means of the FPGAs is called `FirmWare`.
FPGAs offer the possibility to implement glue logic depending on the application

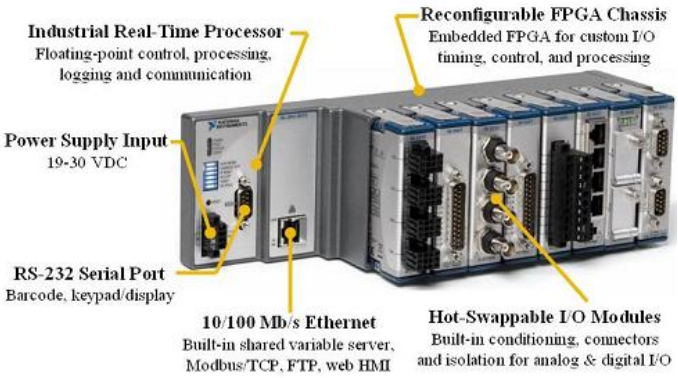



Figure 1.3. National Instruments CompactRIO picture taken from the National Instruments web site

but, also, to implement complex DSP-like algorithms and even embedded processors. A good introduction to these aspects can be seen in [16].
FPGAs are often used as software acceleration in reconfigurable computers (RC), in which the operating system is a standard off-the-shelf real time operating system such as Linux and VxWorks; at university of California at Berkeley has been recently developed a custom FPGA operating system for reconfigurable computers called `BORPH` [9].
As mentioned before the FPGA devices can host soft core processors on which can run standard or custom operating systems. Some examples can be seen in [2] [21] [17] [7] and [23].
The object of this work is to develop a hardware operating system for mechatronic applications, which means that the FPGA device does not host a soft core processor, able to execute one only operation at a time, but it executes many concurrent hard real time functions allowing the user to develop his own application code taking advantage of the main features of the device: concurrency, flexibility and determinism.

## 1.2 FPGA Architecture

The FPGA (Field-Programmable Gate Array) are semiconductor devices containing programmable components called `Logic Elements` (LE) or `Logic Blocks` (LB) and programmable interconnections between the blocks.
This means that the designer does not have to go through the costly and time consuming process of manufacturing an ASIC (Application Specific Integrated Circuit). The major benefit with the FPGA is the possibility to reconfigure the device if the design changes. Even the pin assignment can be changed to some extent if it turns

out to be a Printed Circuit Board (PCB) design error.
Main disadvantages with respect to ASIC solution are:

- FPGAs are slower
- FPGAs handle simpler designs
- FPGAs are more power consuming

Nevertheless FPGAs are often used as first phase of an ASIC design.
A classic FPGA logic element consists in a 6(or 8)-input lookup table (LUT) and a flip-flop.
There is only one output, which can either be the registered or the unregistered LUT output. The logic block has six or eight inputs for the LUT and a clock input. Since clock signals, and often other high-fanout signals, are normally routed via dedicated routing (global network), they and other signals are separately managed.
To define the behavior of the FPGA the user provides a hardware description language (HDL) or a schematic design. Common HDLs are `VHDL` and `Verilog`. Then a technology-mapped `netlist` must be generated. The `netlist` can then be fitted to the actual FPGA architecture using a process called place-and-route. The user will validate the map, place and route results via timing analysis, simulation, and other verification methodologies. Once the design and validation process is complete, the binary file generated is used to program the FPGA.
Commonly the code designed and downloaded on an FPGA is called firmware and its parts are called IP-cores or IP-blocks (Intellectual Properties cores/blocks).
All these passages are usually performed by means of a software tool provided by the device manufacturer.
In addition to logic elements, FPGAs provides I/O pins, memory and clock management.
To interconnect the device to the external world, the I/O pins are configurable between different standards (LVTTL, LVPECL, LVDS and so on) with the possibility to set pull-up/down resistors, terminations, slew rate and current strength.
Memory is usually divided into distributed memory and block memory. Distributed memory consists in the LUTs of the logic elements; using this memory more area is spent but the application runs faster.
Block memory is located in a defined location of the chip and is divided into blocks of some kilo bits. The total amount of RAM depends on the FPGA family, size and producer but, up to now it is always less than a few Mbits .

## 1.3 Real Time Systems

Usually an operating system is intended as a platform on which programs, or better, processes, can be executed. An operating system is also a resources allocator.
With the FPGA we can not speak about programs and resources as we are used to do while we can use the word `process` to indicate a portion of the code, or algorithm, implemented on the chip.
First of all the language used to write an FPGA code is not a programming language but a hardware description language and the output is a low level logic algorithm and many of the notions used for the computer world operating systems are not applicable.
For sure it is possible to give some definitions of real time systems and real time applications that become constraints and specification for a real time operating system.
There is not only one approved definition of real time application but the basic idea is that a real time system is called so when the overall correctness depends on both the functional correctness and timing correctness.
In other words, it means that a real time operating system has to allow executing tasks with strictly time constraints called deadlines.
A distinction has been made between soft real time systems and hard real time systems and the main difference is:

- In a soft real time system the deadlines must be `generally` met

while

- In a hard real time system the deadlines must be `deterministically` met

In this way consider a system, soft or hard real time, depends from the application and from how huge are the damages in case of deadline missing.
The main advantage of the FPGAs with respect to processors is the possibility to execute different concurrent processes but the FPGA operating system concept is often applied to an operating system, custom or standard, running on a soft core which implements a processor.
A soft core processor on an FPGA allows to execute code written in common programming language like C or C++ with the advantage of custom hardware acceleration performed building custom complex instructions directly on the FPGA logic. However, only one instruction is executed at a time and the FPGA become a sequential processor.
If the area on the FPGA is large enough, it is possible to instantiate more than one soft core processor in a single chip allowing the execution in parallel of different tasks.

In my opinion, this is a great limitation for the FPGA potential because the parallelism degree is always limited and a great FPGA has a cost higher than the sum of the costs of standard discrete processors.
Furthermore the scheduler implemented on the soft core processor, which allows multi task operations, has always a system overhead due to the context switching that is the change of context due to an interrupt request.
The context switching overhead is the main reason for deadline missing when the frequency of the service routine increases because the time lost while saving and restoring the context variables become non negligible with respect to the time between two consecutive interrupts.
For this reason my work was concentrated on creating a firmware framework for FPGA which allows the application designer to create and instantiate its own IP-cores disregarding the hardware device on which they will be executed, that is the main purpose of a generic operating system. In section 2 this concept will be explained more extensively.

# Chapter 2

# Digital Programmable Platform for Mechatronics

The environment in which the project has been developed is the mecatronics where mechanic engineering, electronic engineering and software engineering are gathered in order to create a complete system, control it and interface it with the external world.
This system can be simply represented with the scheme 2.1 that we called `mechatronic framework` which follows the PKH (Plant Control Host) paradigm.

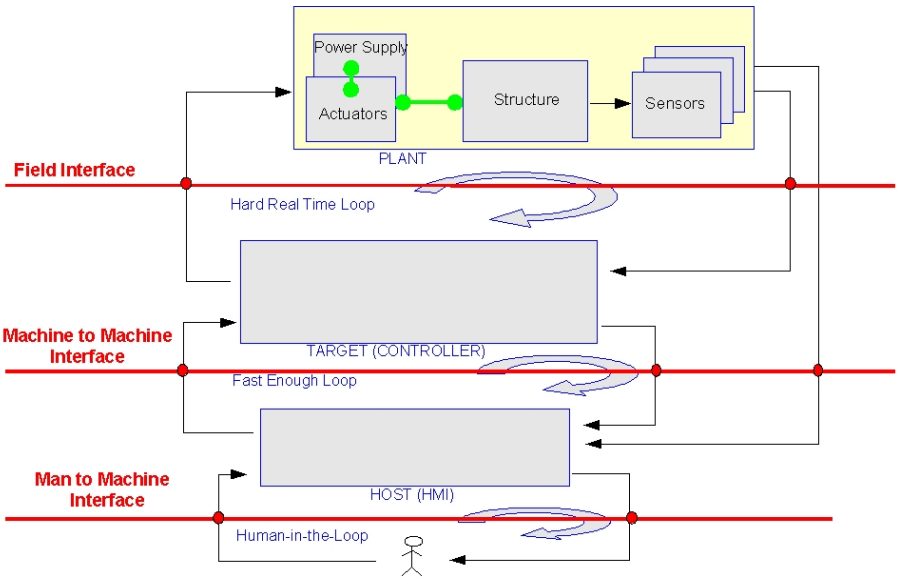



Figure 2.1. Mechatronic Framework

Starting from the top it is possible to see the Plant which is composed of:

- Power Supply which exchanges energy with the system through the actuators
- Actuators which transform, normally, electrical energy into mechanical energy
- Mechanical structure that is the object to move and to control
- Sensors which allows the controller or the host to have a feedback from the field

The second layer is the controller also called `target` from the point of view of the host. Plant and controller communicate by means of the Field Interface divided into commands and sensors signals.
From the control point of view, plant and controller are inserted in a Hard Real Time loop.
The third layer is the Host, usually a PC which acts as the system supervisor and allows the user to set control parameters and acquire real time variables.
Controller and Host are inserted in a Fast Enough loop. This loop can not be defined as real time because, usually, an non real time operating system runs on the host PC, and the communication layer (Machine to Machine Interface) can not have real time characteristics.
The last layer on the bottom represents the user who is often present especially in a prototyping phase when he becomes a part of the loop modifying the parameters of the control.
The object of this work is the controller and especially the firmware part of it.
The main idea was to develop a cross platform operating system that can be used on different digital platforms which have a similar architecture.
For this reason, we started working with an electronic control unit developed in our laboratory. Later, others versions of this ECU has been developed and finally born a spin-off company (ACTUA S.R.l.) that produces these boards with the name of `EKU` (Electronic Kontrol Unit).
The architecture of these boards was always the same and, even if the components change between a version to the other, the kernel of the operating system was kept.
The last version of the EKU is shown in the pictures 2.2 and 2.3 where key components and connectors are evidenced.

## 2.1 EKU Kernel

This board kernel is based on two main programmable devices:

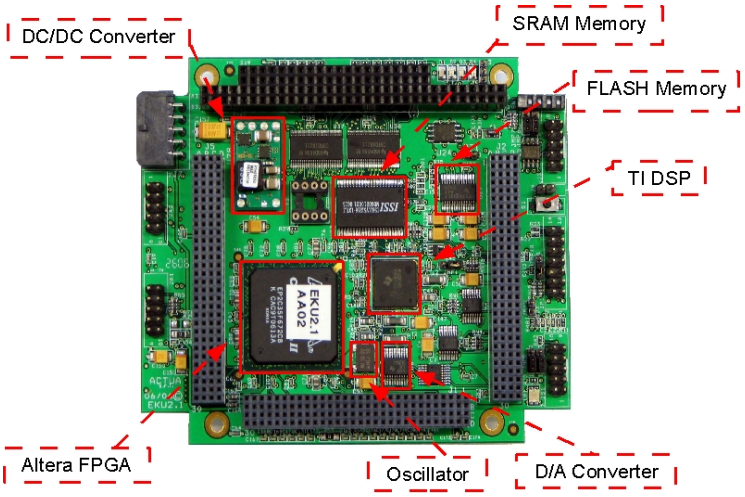



Figure 2.2. `EKU 2.1`: key components

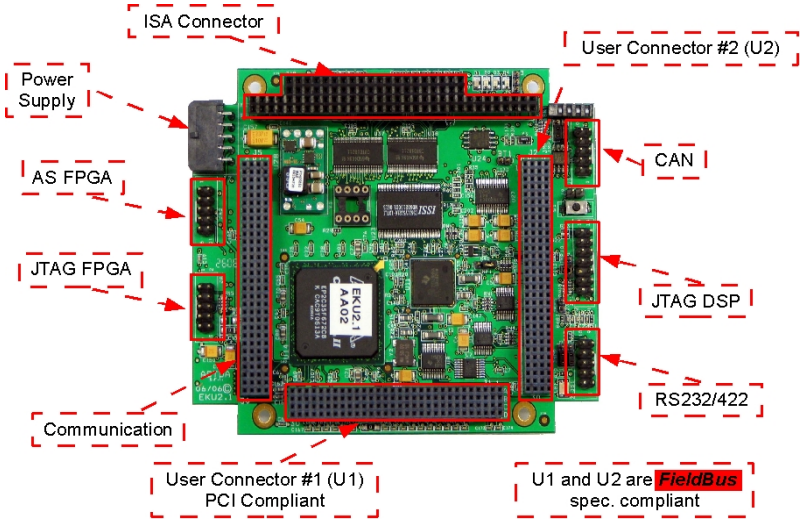



Figure 2.3. `EKU 2.1`: connectors

- DSP (Digital Signal Processor) (TMS320F2812) from Texas Instruments; is a fixed point 32 bits DSP with an external memory interface of 16 bits data width and peripherals targeted for automation control (I/O captures, PWM signals, Encoder interface, 16 channels ADC and more)

- FPGA (Field Programmable Gate Array) (CycloneII EP2C35F672C8) from ALTERA; it has about 35000 logic elements, 480 kbits of RAM, different I/O

standards and four PLLs (Phase Locked Loop) for clock managements

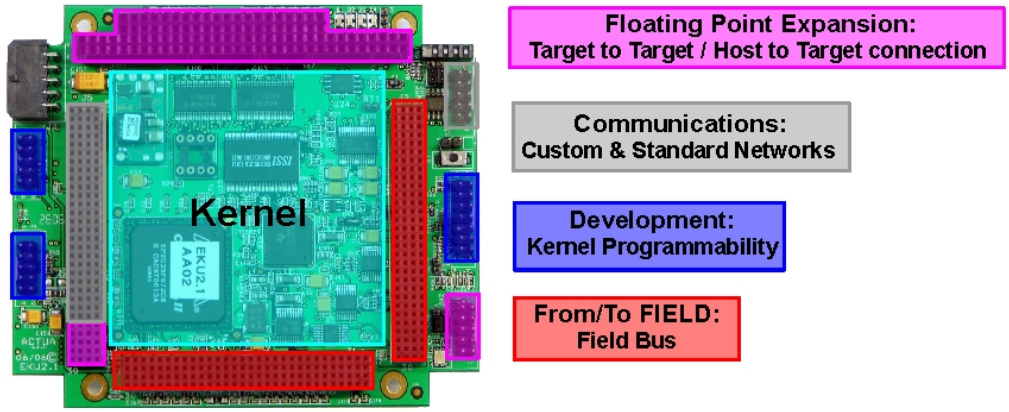



Figure 2.4. EKU 2.1: Kernel and Interfaces

The main idea of the Hard Real Time Operating System for our digital platforms is illustrated in figure 2.5 in which the concept mentioned in 1.3 for both the firmware (FPGA device) and the software (DSP device) is shown.

HRTOS is intended as an abstraction layer that allows the user to program his

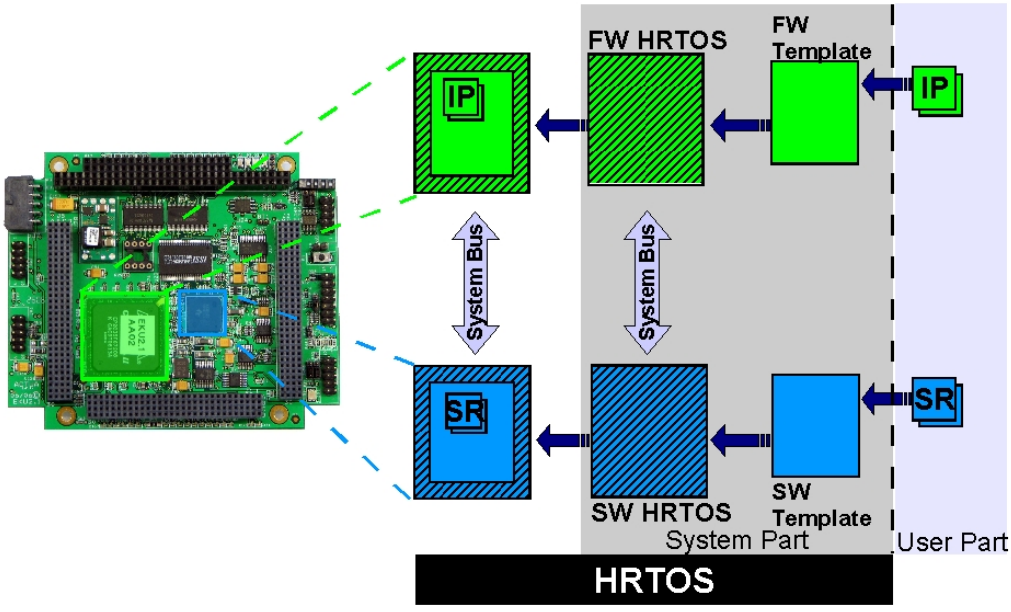



Figure 2.5. HRTOS main idea

code independently from the hardware devices and the connections with them.
The operating system gives the rules to the user in order to respect the physical constraints related to the available resources and is customized and compiled by the system analyst for the specific digital platform.
The user code are called SR (Service Routines) when related to the software and IP (Intellectual Properties) when related to the firmware.
SRs and IPs are linked to the HRTOS by means of a software template and a firmware template which are user friendly framework containing the links to all the resources available on the device. In these templates the user instantiates his own code divided in more SRs and IPs and links its interfaces to those of the template in a logical way, neglecting the physical aspects.
SRs and IPs communicate by means of the HRTOS and precisely through the System Bus that is a deterministic protocol based on the memory mapping concept.
The system bus uses the external memory interface of the DSP in order to create a single master (DSP) single salve (FPGA) communication channel able to exchange up to 16 bits in a single transaction.
The determinism is guaranteed by the use of a signal, managed by the slave, able to hold the master until the write or read operation has been completely performed. In this way the duration of the transaction is defined `a priori` by the time this signal is held active.
The system bus includes the possibility to interrupt the execution of the code running on the DSP by means of 16 interrupt lines multiplexed on two DSP hardware interrupts. More details on the system bus can be seen in chapter 3.

## 2.2 EKU Interfaces

The EKU 2.1 interfaces are divided into four parts (see figure 2.4):

- Floating Point Expansion which allows to interconnect the EKU to a real time micro processor in order to create a target to target communication for co-processing designs, or to a host PC for data logging, monitoring and tuning

- Communications which allow to create a custom hard real time grid between two or more EKUs for distributed systems that require high computational power, or to interconnect the EKU with the external `world` by means of standard automotive or industrial protocols

- Development that is the kernel programmability interface (JTAG connections for DSP and FPGA)

- From/To Field that implements the field bus used to interconnect the EKU with digital and analog sensors and peripherals, used to interconnect the EKU with one or more field modules, and used to interconnect the EKU with analog and digital signal power supplies

### 2.2.1 Floating Point Expansion

The Floating Point expansion consists in a communication channel between the EKU and a micro processor, usually a PC.
As mentioned before this communication channel has two main purposes: real time target to target communication for co-processing designs, and non real time host to target communication for data acquisitions, monitoring and tuning.
The EKU has two possibilities to perform this channel:

- with a serial RS232/RS422 channel only for host to target communications
- through ISA (Industry Standard Architecture) protocol for both target to target and host to target communications

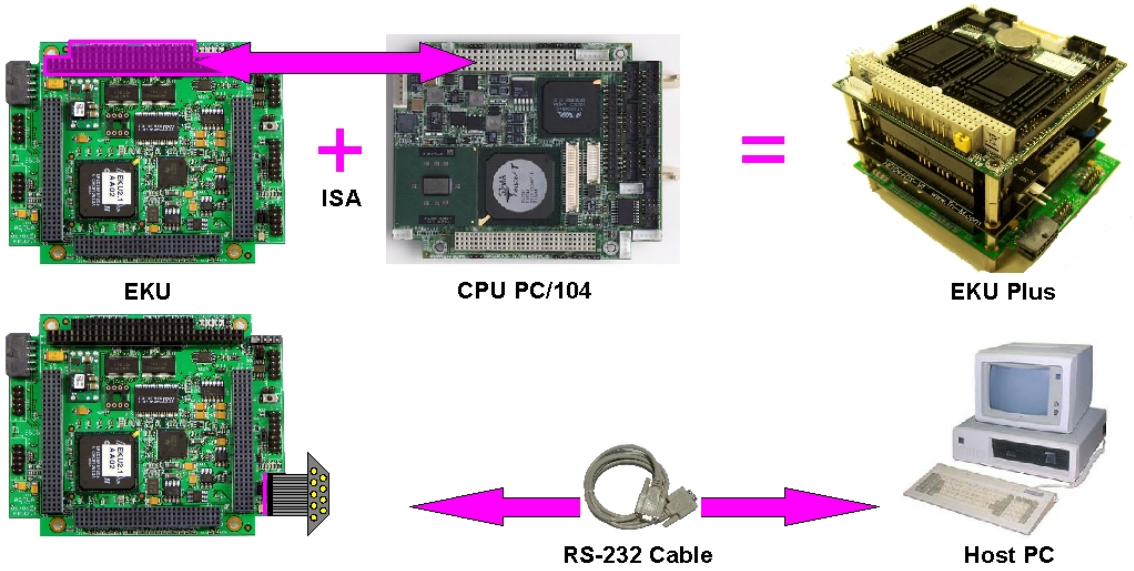



Figure 2.6. Floating Point Expansion: EKU and PC through ISA Potocol (top); EKU and PC through RS232/422 protocol (bottom)

These configurations are shown in figure 2.6 where the top part illustrates the stack composed of an EKU and a PC, both compliant to the standard pc/104; this configuration is called `EKUplus` and the link between the EKU and the PC is also

called `HRT Tandem` because it is possible to perform a Hard Real Time communication between the fixed point DSP, the FPGA and the floating point micro processor using the ISA protocol.
While the RS232/RS422 protocol is managed by the DSP, the ISA protocol is entirely managed by the FPGA with an operating system dedicated IP. This IP is explained better in section 4.1.

### 2.2.2 Communications

The Communication Interface is intended to manage both standard communication protocols (usually to create standard networks) and a custom communication protocol developed in order to create a hard real time grid (HRT Grid) between different EKUs in the same application.

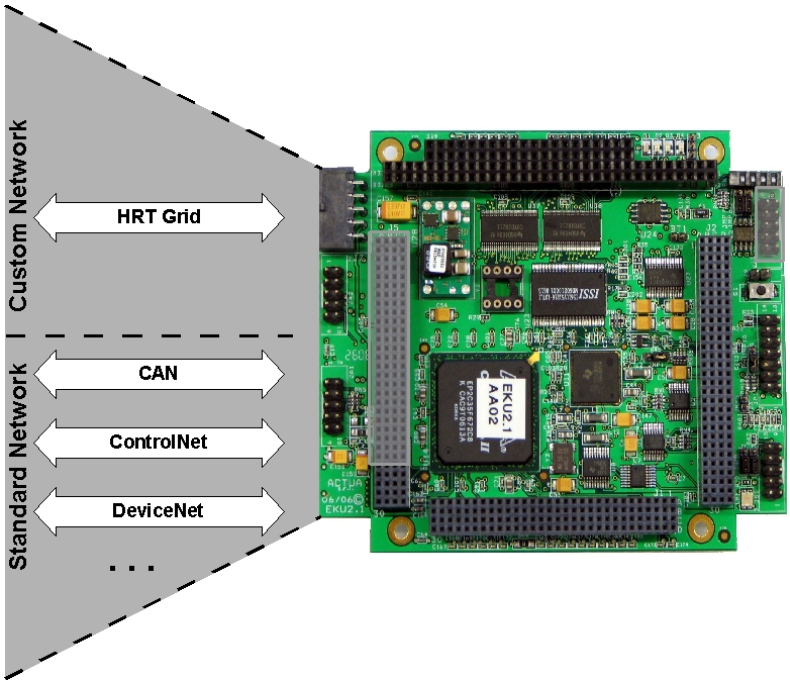



Figure 2.7. Communication Interface

Standard network protocol can be, for instance, the CAN bus protocol that can be managed directly by the DSP which has a native CAN controller, or external CAN bus channels implemented on a separated board (dCANi_AnyB 1.1) developed in order to have two more CAN channels electrically isolated (see figure 2.8).

With the same board other protocols can be used because there is a connector

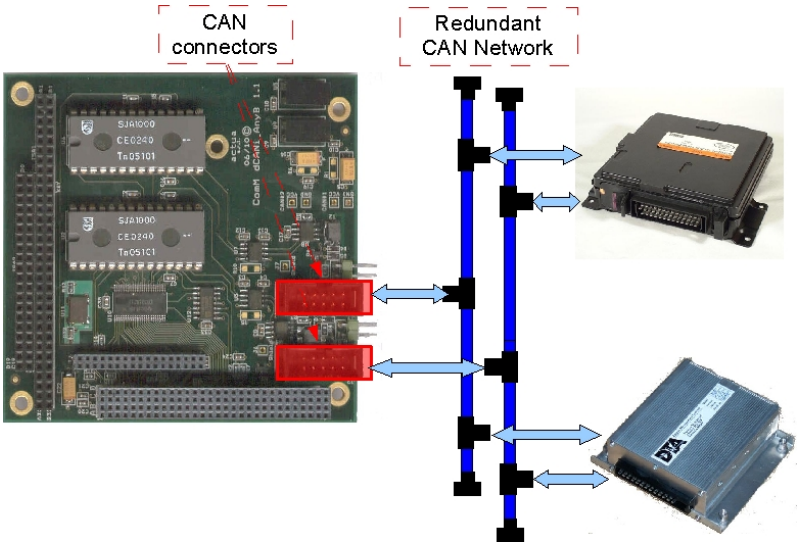



Figure 2.8. dCANi_anyB1.1 : Dual isolated CAN bus board used for redundancy

on the board that is compliant with the anybus slave modules, which performs a lot of standard industrial and automotive protocols such as ControlNet, DeviceNet, CANopen, Modbus, Profinet-IO, Profibus and so on (see figure 2.9).

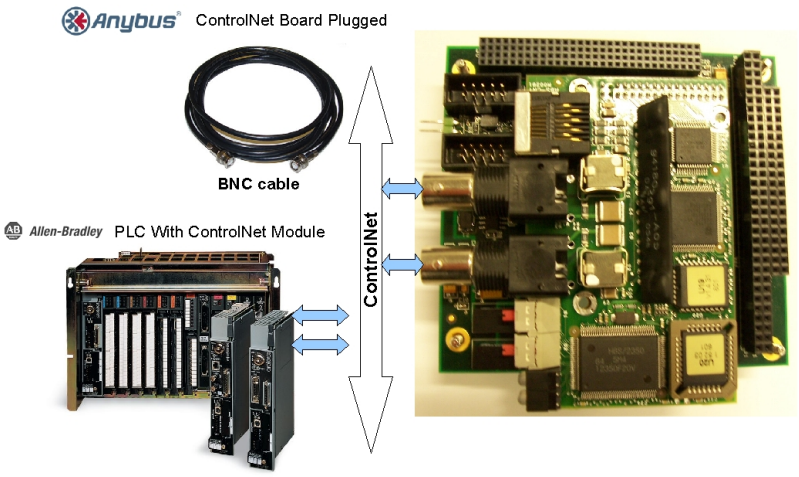



Figure 2.9. dCANi_anyB board used as bridge for anybus modules in standard networks

While the higher level of these protocols is managed by external devices, the communication between the external devices and the DSP is performed by the FPGA that creates a link between the DSP external memory interface and the registers of the chips. In this way the DSP user can easily access this devices as physically mapped in the DSP accessible memory.
The HRT Grid is a custom protocol, entirely managed by the FPGA, which allows a high performance communication bus between EKUs. This bus is used for remote task synchronization and few data exchange in scheduled periods opposite to standard networks which are used to transfer a large amount of data as fast as possible. More details on the communication interface can be seen in section 4.2.

### 2.2.3 From/To Field

The From/To Field Interface is divided between DSP and FPGA; DSP can manage both analog and digital signal since it has integrated a 16 channel analog to digital converter, while the FPGA can directly manage only digital signals.
Neither DSP nor FPGA has an integrated DAC device and to overcome this limitation an external DAC device has been provided on the EKU. This device is controlled by a dedicated IP-core of the firmware operating system. More details on the field interface are in section 4.3.

# Chapter 3

# System Bus

## 3.1 Memory Mapping

As mentioned in the previous chapter, the System Bus is based on the DSP external memory interface.
The DSP used in the digital platform has a 16 bits wide data bus compliant with the AMI (Asynchronous Memory Interface) standard.
In addition to the data bus, the AMI protocol implemented in the DSP has an address bus of 20 bits able to address up to one mega locations of 16 bits each and few control signals such as:

- three chip select signals able to address different regions with different parameters
- RnW signal that is high when the transaction is a write and low when it is a read
- WEn signal (active low) indicates a write operation
- RDn signal (active low) indicates a read operation
- Hold signal (active low) is used by the external device to hold the DSP until the transaction has been fully performed

The AMI protocol is a single master single slave protocol that performs only single word transactions (no burst transfer is allowed) and the DSP is always the master of the channel while the FPGA is the slave.
In order to address more than one IP internally in the FPGA, the bus is split into many interfaces each of them activated when the DSP address is in a precise range.
The memory regions addressed by the DSP, by means of the three chip select signals, are shown in figure 3.1; the regions six and seven has been reserved for DSP code

and data, while the regions zero, one and two are used by the firmware operating system to map its IPs.

In this way the code running on the DSP can easily access the registers and

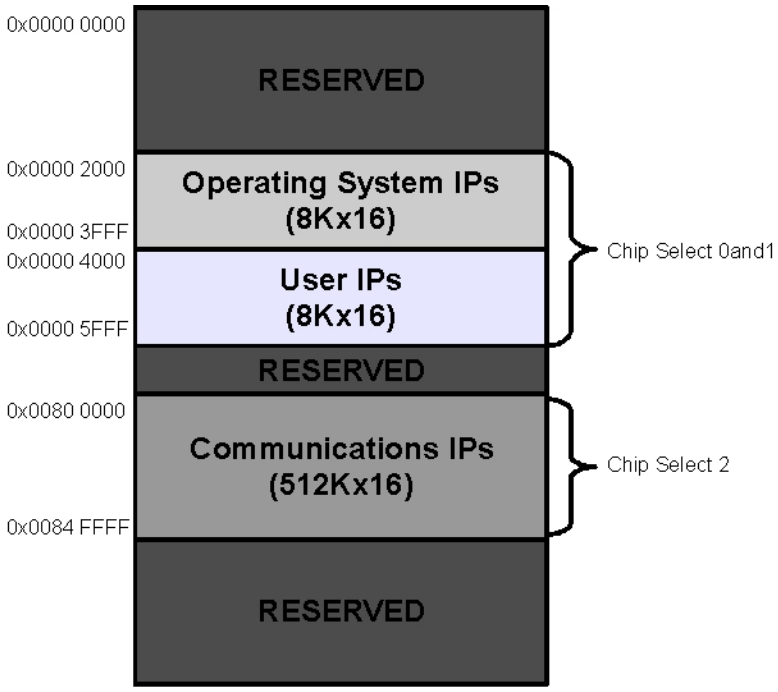



Figure 3.1. DSP-FPGA Memory Mapping

functions implemented on the FPGA by means of a memory pointer, in the same way as it accesses the resources embedded on the DSP.
This technique is widely used in the computer field and it is known as memory mapping.
The first chip select (chip select 0and1) addresses two memory zones:

- In zone 0 are mapped the IPs of the operating system such as the DAC management IP and the Interrupts manager IPs
- In zone 1 are mapped the IPs written by the FPGA user (application designer)

The system IPs are statically memory mapped because this operation needs to be executed once, while the user IPs are dynamically mapped starting from the address 0x0000 4000.
The second chip select (chip select 2) addresses the zone 2 in which are mapped the IPs related to the communications such as the CAN interface (that is a bridge between the DSP and the SJA1000 device used on the `dCANi_anyB 1.1`, the Anybus

interface between the DSP and the anybus modules, the ISA protocol to communicate with the PC and the HRT Grid protocol used to create a grid of EKUs.
The AMI protocol is very simple but it is not very common in FPGA environment. For this reason inside the operating system the IPs use a different protocol that is called `wishbone protocol`.
The wishbone protocol is again an asynchronous protocol which uses:

- two data buses (data_in and data_out)
- address bus
- strobe signal (active high) that indicates when a transaction is being performed
- WE signal (active high) that indicates when the transaction is a write operation
- ACK signal (active high) that is asserted by the slave to indicate that the operation is completed

The wishbone protocol allows many types of transaction included the burst mode but since the AMI protocol allows only single transactions, only this mode has been implemented.
The translation from the AMI protocol into the Wishbone one is performed simply with combinational logic:

- wishbone data input bus is fed by the DSP data bus
- DSP data output bus is fed by the wishbone data output only when the operation is a read
- strobe signal is a combination of the address bus and the chip selects
- DSP hold signal is fed by the ACK signal of the wishbone protocol

To ensure that the DSP is released when the transaction is really completed, the hold signal is de-asserted after the ACK signal is gone high and returns low (after the strobe signal goes low).
The logic of the hold and ACK signals is synchronous and takes on some clock periods to be performed. The entire transaction of a word between the DSP and the FPGA takes on about $160nS$ both for read and write operations.

## 3.2 Address Decoder

In each zone addressed by the DSP through the system bus are allocated more than one IP-core. It means that an address decoder is needed in order to choose which of these IP-cores must respond.
The first layer of the address decoder is performed directly by the IP translating the AMI protocol into the wishbone one using the chip select signals and the bit number 14 of the address bus. In this way, the three zones (0, 1 and 2) are correctly recognized: the zone 2 directly by the relative chip select, while zone 0 if the 14th address bit is equal to zero and zone 1 vice versa with the chip select 0and1.
Every IP-core mapped in a single region can have different number of registers and so it can take up more or less memory locations than another one.
For this reason a new layer for the address decoder is needed. In zones 0 are mapped the IP-cores listed below:

- System Config IP (start address 0x0000 2000) takes up 16 registers
- User Interrupt Manager IP (start address 0x0000 2010) takes up 8 registers
- System Interrupt Manager IP (start address 0x0000 2018) takes up 8 registers
- DAC Manager IP (start address 0x0000 2020) takes up 16 registers

The System Config IP is a configuration module used to set some parameters of the operating system and to read the status of some registers.
It acts also as a time keeper counting, with a precision of a milliseconds the time since the EKU has been powered up.
For the time keeper function three registers are used for a total of 42 bits (10 bits for milliseconds, 6 bits for seconds, 6 bits for minutes and 20 bits for hours) able to count for longer than 19 years.
The User Interrupt Manager is used to interrupt the execution of the DSP code when a particular event is recognized by one of the user IP-cores. For example, a reason to interrupt the DSP can be a particular angle reached by an electrical motor in order to execute angular sampling and control; or else the output of an hardware comparator can be connected to an FPGA pin and then linked, after a digital filtering, to an interrupt request line to interrupt the DSP control algorithm when an analog signal reaches a predefined threshold.
The System Interrupt Manager is used by some of the operating system IPs to interrupt the execution of the DSP code and to trigger particular tasks related to the IPs themselves.
More details about these IP-cores are given in the next section.
The DAC Manager IP is used to configure and use an external multi channel DAC device. The IP performs the configuration during the initialization of the system

and then feeds the inputs of the DAC during run time. More details about this IP are given in section 4.3.

In zone 1, the user IPs instantiates are dynamically mapped by the firmware developer in the operating system template.
In order to correctly map these IPs, the user must set a constant named `NUM_IP_CONST` with the number of the IPs he wants to instantiate in the system.
Each user IP has 16 register locations available; in this way it is simple for the operating system to assign memory ranges to the IPs starting from the address 0x0000 4000.
For every IP instantiated by the user, a number (`num_ip`) that is used by the operating system is declared to order the IPs in the memory zone. The system can host up to 64 IPs; for example if three user IPs are used, one performs the read of an incremental encoder and the hall effect sensors with num_ip = 0, the second is used to control the current in a DC brush-less motor with num_ip = 2, and the last one performs a PWM signals modulation with num_ip = 1. Finally, the memory mapping will be:

- Encoder IP (start address 0x0000 4000)
- PWM modulator IP (start address 0x0000 4010)
- Current control IP (start address 0x0000 4020)

To write on the registers unused of the zone 1 has no effects, while the reads return the value 0x0000. The ACK signal of the wishbone protocol is fed with the strobe signal in order to release the DSP as soon as possible.

In zone 2, the system IPs related with the communication with the external world are mapped and a particular IP of the system bus is used to implement memory sharing between DSP and FPGA.
The order implemented in this portion of the address decoder is listed below:

- Anybus modules IP (start address 0x0008 0000) takes up 4096 registers
- CAN bus channel 1 IP (start address 0x0008 1000) takes up 128 registers
- CAN bus channel 2 IP (start address 0x0008 1080) takes up 128 registers
- RSMP channel 1 (RS multi protocol) IP (start address 0x0008 1100) takes up 16 registers
- RSMP channel 2 (RS multi protocol) IP (start address 0x0008 1110) takes up 16 registers

- RSMP channel 3 (RS multi protocol) IP (start address 0x0008 1120) takes up 16 registers
- RSMP channel 4 (RS multi protocol) IP (start address 0x0008 1130) takes up 16 registers
- Scheduler 1 IP (start address 0x0008 1200) takes up 32 registers
- Scheduler 2 IP (start address 0x0008 1220) takes up 32 registers
- Scheduler 3 IP (start address 0x0008 1240) takes up 32 registers
- Scheduler 4 IP (start address 0x0008 1260) takes up 32 registers
- HRT Grid config IP (start address 0x0008 1280) takes up 32 registers
- HRT Grid registers IP (start address 0x0008 12A0) takes up 32 registers
- ISA core IP (start address 0x0008 2000) takes up 8192 registers
- DSP2FPGA synchronous memory IP (start address 0x0008 4000) takes up 4096 registers

Anybus modules IP and CAN bus IPs are simply bridges between the system bus and the interface of the external devices.
The DSP can easily access these devices as for all the others IPs instantiated in the FPGA through a memory pointer.

RSMP channels IPs are system IPs that implement the standard RS protocol such as RS232, RS422 and RS485. The host interface of these IPs is the standard UART and are used with the board developed by ACTUA S.r.l called `qRSMPi`, which stands for quad RS multi protocol isolated. This board implements four isolated RS standard communication channels.

Scheduler IPs have been developed in order to trigger the DSP service routines starting from events generated into or acquired by the FPGA.
These schedulers are directly connected to the HRT Grid IP and allow to trigger three different service routines for each DSP in the grid; it is possible to trigger the same service routines on all the EKUs composing the grid or to trigger, for example, the service routine number two on a DSP and the service routine number one on another DSP.
These scheduler IPs perform the so called `Distributed Scheduler` discussed deeper in section 4.2.

HRT Grid config IP is used to set parameters of the grid configuration such as the number of boards in the grid, the identification number of the board inside the grid, which are the boards enabled to write on the bus and others.

The HRT Grid registers IP contains the registers shared in the grid, since the main feature of the HRT Grid is the shared memory between the boards.
These 32 registers are divided into two parts: registers written by the DSPs and registers written by the FPGAs. DSPs usually shares control variables of the running code while FPGAs normally share services such as an encoder sensor measurement, but there are not limitations due to the implementation.
HRT Grid config and HRT Grid registers IPs are explained better in section 4.2.

ISA core IP allows to communicate with a floating point PC using the Industry Standard Architecture protocol. This protocol is still largely used in industry environment because of its low price and it has enough throughput for many control applications.
The communication between the EKU and the PC is based on the concept of the shared memory; in this way DSP and PC can perform a co-processing application in the easiest way: using the same variables and parameters and addressing them by means of a simple memory pointer.
PC and DSP can trigger a service routine to each other simply writing on a dedicated register.
More details on this IP are given in section 4.1.

DSP2FPGA synchronous memory IP allows to exchange consistent patterns of data between DSP and FPGA. This can be very useful when more than one control variable has to be passed from the DSP to the FPGA and vice versa, for example to update the coefficients of a filter; in this case a data inconsistency between the coefficients produces errors in the output.
This IP uses some control registers to perform the update of the patterns. The patterns are eight of 256 words each.
More details are given later in this chapter.

These three address decoder perform the second layer of the complete address decoding; the third, and last, layer is performed in each IP, both system IP and user IP.
In order to be able to communicate through the wishbone protocol, every IP must contain a state machine implementing the wishbone slave interface and one of the task of this process is to discern between the register to update (DSP write operation) or to output (DSP read operation) basing on the address in its own IP address range.

Summing up the address decoding function is hierarchical distributed on three layers:

- Layer 1 - AMI to Wishbone protocols translation IP
- Layer 2 - three address decoder IPs, one for each memory zones
- Layer 3 - Each IP in the system has an internal address decoder working on the number of registers it takes up

## 3.3 Interrupt Management

The task of User Interrupt Manager IP is to assert the DSP interrupt request line XINT2 when one or more interrupt request lines (USR_INTn) are asserted by one or more user IP.
In the template of the firmware operating system it is possible to wire the signals of the user IP-cores to the 8 inputs of the interrupt manager IP. In this way a portion of the user code can interrupt the execution of the DSP code in order to start the execution of a specified task associated with that interrupt.
Considering the system as a co-processing between a DSP and an FPGA, the interruption of the DSP code by the FPGA can be seen as a distributed scheduler: in addition to the service routines scheduled by the software operating system, others that are scheduled by the FPGA can be executed at any time. In fact, they usually depend on external events.
The 8 interrupts available in the FPGA template are then multiplexed in a single interrupt request line to the DSP side (see figure 3.2).
One of the registers involved in the interrupt management is the configuration reg-

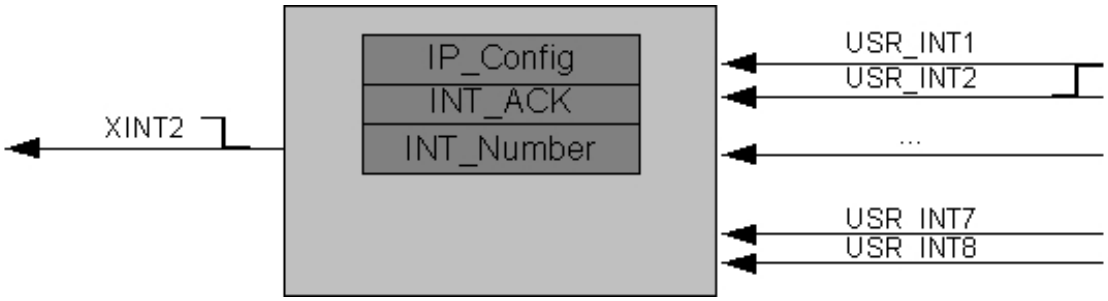



Figure 3.2. User Interrupt Manager: 8 FPGA lines are multiplexed to one DSP hardware interrupt

ister by means of which it is possible to set the event, associated to each interrupt

line, that will trigger the interruption. For each interrupt line are used two bits for a total of 16 bits.
The possibilities are:

- (00) the line is disabled
- (01) the line is active on rising edge
- (10) the line is active on falling edge
- (11) the line is active on both rising and falling edges

A second register reports the number of the (first) line that triggered the interrupt request. In this way, when the DSP scheduler receives the interrupt request, it performs a read operation of this register and decides which is the task to call. This operation is the demultiplexing of the single interrupt line to the original eight. This operation takes some DSP clock cycles but allows to multiply the possibility to interrupt the DSP code.
When the correct task has been performed the DSP scheduler writes it on another register of the interrupt manager which deactivates the request line (XINT2) and starts again listening to the eight inputs.
The System Interrupt Manager is identical to the user one but the eight inputs lines are fed by operating system IPs. The hardware DSP interrupt line is XINT1 instead of XINT2 but the FPGA registers and DSP scheduler have the same behavior.
The associations between the inputs of the interrupt manager and the system IPs are:

- line 0 is associated with the HRT Grid ISR 1 (Interrupt Service Routine 1)
- line 1 is associated with the HRT Grid ISR 2
- line 2 is associated with the HRT Grid ISR 3
- line 3 is associated with the HRT Grid Fault ISR
- line 4 is associated with the Interrupt from the PC (through ISA bus)
- line 5 is associated with the Anybus module ISR
- line 6 is associated with the ISR of the first CAN channel
- line 7 is associated with the ISR of the second CAN channel

## 3.4 DSP-FPGA Synchronous Memory

As mentioned before, this IP is intended to perform a block data exchange between DSP and FPGA in a consistent way. It means that writes and reads on the same block can not be interposed.
Being the DSP and the FPGA two autonomous devices, the probability of data inconsistency is high; however, in a hard real time system even a low probability can not be accepted.
For this reason a particular IP-core has been developed to help the application designer to avoid this problem.
The principle at the bottom of this IP is the concept of the dual port memory.
In a dual port memory, read operation during a write operation is allowed: two distinct data and address buses are provided and they can be used by two different devices. However the limitation of the data consistency is not solved with a simple dual port memory. To overcome this problem two dual port memory blocks must be used.
A simplified block scheme of this idea is shown in figure 3.3 where signals indicated

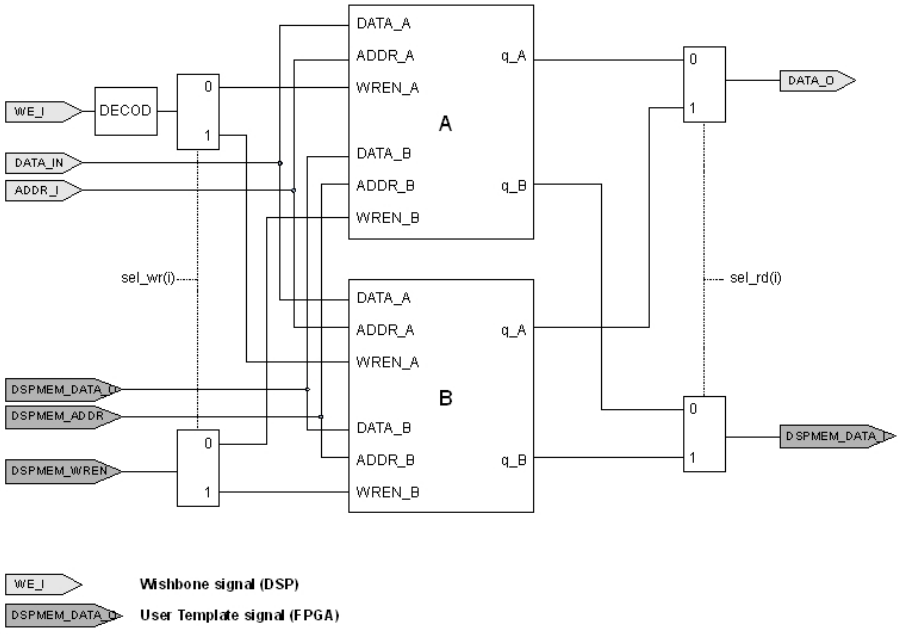



Figure 3.3. Structure of the double dual port memory at the base of the IP

as `Wishbone signal (DSP)` are the signals of the system bus (driven by the DSP) and then translated to the wishbone protocol, while the signals indicated as `User Template signal (FPGA)` are driven by the user IPs instantiated in the firmware

user template.
While the first block is updated by a device that writes values on it, the second device can read from the second block and vice versa. At the end of the writing phase, while the second device is not performing read operations, the pointers to the memory blocks are exchanged. Then the first device writes on the second block while the second device reads from the first one.
To understand better this mechanism an example is proposed: the DSP code has to transfer to the FPGA code a list of parameters for the position profile of an AC brush-less motor. The FPGA code needs the complete set of these parameters in order to perform a linear interpolation to obtain the position profile as a function of time.
The designer chooses the fourth portion of memory among the eight available; at the beginning, the memory is empty and the signal called `ready` (active high) is low. The DSP code starts filling the block A of the fourth memory portion and when it has finished, it sets a configuration register of the IP accordingly. The register field is called `swap_en`.
At this point, the IP changes the writing pointer from the block A to the block B while the read pointer remains set to block A and the `ready` signal becomes high (active).
The FPGA code can start reading parameters from the block A. Then a signal called `busy` is emitted from the FPGA to indicate that the memory block can not be updated or the data consistency can not be assured.
While the FPGA code is reading from the block A, the DSP code has a new set of parameters to write, which is done on block B. When the DSP code has finished to fill the memory, it sets again the `swap_en` field which remains active until the FPGA code has not finished to read the parameters from the block A. At this point the `busy` signal is de-asserted and the IP changes the writing pointer from block B to block A and the reading pointer from block A to block B.
Now the DSP code can start again writing on block A while the FPGA code reads the new parameters from block B.
Obviously these operations can be performed also in the other direction: the FPGA code writes data and the DSP code reads but the procedure is substantially the same.
A consideration must be done: as two dual port memory blocks (A and B) for each portion of memory are used, the RAM bits used is doubled with respect to the useful memory bits. However, this choice has been motivated by the will to maximize the flexibility.

# Chapter 4

# System Interfaces

This chapter presents the FPGA operating system implementation of the EKU interfaces.
As seen in chapter 2 the main interfaces of our digital programmable platform are:

- Floating point expansion (EKU-PC via ISA)
- Communications (standard networks and custom HRT Grid)
- From/To Field (Field bus)

Floating point expansion is implemented in a single IP-core called ISA core performing two communication channels:

- Target to target communication (EKU-PC co-processing)
- Host to target communication (data logging, monitoring and tuning)

Concerning Field interface, the FPGA field interface and how the FPGA operating system manages this interface will be discussed. The DSP field interface is the object of this thesis.

In the Communication interface a distinction will be done between standard communication protocols (networks) and a custom communication protocol (HRT Grid). Standard networks are managed by external devices; the FPGA task is to allow the interaction with these devices.
HRT Grid is entirely managed by the FPGA and main features and characteristics are presented.

## 4.1 Floating Point Expansion

With floating point expansion we intend the integration in the system of a floating point micro processor. This floating point micro processor can be seen either as a co-processor for hard real time applications, or as a host PC able to acquire from the target and present to the user a large amount of data (real time variables). In this case the PC can also be a bridge between the target and a standard network (for example a LAN) in order to act on the EKU remotely.
The link between EKU and PC is the ISA protocol (Industry Standard Architecture) since it is still a protocol commonly used in industrial systems and because it guarantees direct access to PC peripherals such as DMA controller (for PC memory direct access), interrupt controller (direct access to the 15 hardware CPU interrupts) and all the other peripherals mapped in low memory region.
ISA is an asynchronous memory interface protocol with separated address and data buses; the data bus is 16bits wide with the possibility to perform 8bits write and read operations.
ISA is a low level PC protocol which allows to access two distinct memory regions (or range): low memory (from 0x0000 to 0xFFFF), called PORT I/O, used to access the PC low level peripherals and the high memory (from 0x00010000), called MEMORY I/O, used as a standard memory.
In our implementation PORT I/O range is used for IP configuration registers while MEMORY I/O range is used as co-processing shared memory.

### 4.1.1 HRT Tandem

As seen in section 2 the digital platform used to develop the system is based on an FPGA and fixed point DSP.
The EKU is an electronic control unit intended for rapid prototyping purpose; for this reason sometimes very complex algorithms have to be implemented and evaluated.
The fixed point mathematics is often precise enough to implement the most control algorithms but writing complex code in fixed point is difficult.
The FPGA can implement operations in single precision floating point and can be used as a DSP mathematical co-processor for simple control codes where the dynamics and precisions require floating point computation. Floating point operations implemented on the FPGA take up a lot of logic elements each and so are limited in number.
The DSP can execute floating point operations but they are emulated since the DSP has not a hardware floating point unit. Time spent by every floating point multiplication is about $2.3\mu S$ while a division takes about $3.2\mu S$, which is ten times slower

than the computation performed in fixed point.
When the DSP and the FPGA are not sufficient to implement the algorithms, the use of a micro processor becomes necessary.
EKU and PC used for real time co-processing by means of a target to target communication have been called `HRT Tandem` to indicate two entities working together (in hard real time) for a common purpose.
HRT Tandem is based on the concept of memory share: DSP, FPGA and PC execute tasks sharing the same variables and parameters. In this way data inconsistency is avoided and the code variables are linked using memory pointers. To implement this architecture a multi port memory is necessary and for this reason the FPGA has been used.
The memory shared between the EKU and the PC is physically located on the FPGA (FPGA RAM blocks) which acts as a switch in order to allow both DSP and microprocessor to access the same memory.
The total shared memory is $8Kbytes$ ($4Kwords$) and is divided into different sections:

- 256 32bits locations shared between PC and DSP
- 256 16bits locations shared between PC and DSP
- 256 32bits locations shared between PC and FPGA
- 256 16bits locations shared between PC and FPGA
- Look Up Table (LUT) of 2048 words (16bits) shared between PC and FPGA

Both system bus (DSP-FPGA) and ISA (PC-FPGA) have a data bus of 16 bits wide; considering to exchange floating point data or even long integer variables, there is the possibility that a 32 bits variable is read while only the least significant word has been written. In this case data inconsistency occurs.
The FPGA, using temporary variables, performs atomically writes into and reads from the 32 bits memory sections; that means these operations are not interrupted by any other ones.
The LUT is filled with discretized trigonometric functions in order to avoid the functions computing, or, moreover, in order to easily generate trigonometric curve profiles simply accessing sequentially the LUT locations and outputting the read values.
The memory mapping, from the PC point of view, is shown in figure 4.1.

The addresses on the right refer to the offset address; it means that these are relative addresses. The absolute address is the sum of the relative one and the base address.
ISA MEMORY I/O range is dynamically allocated: the EKU-ISA driver searches

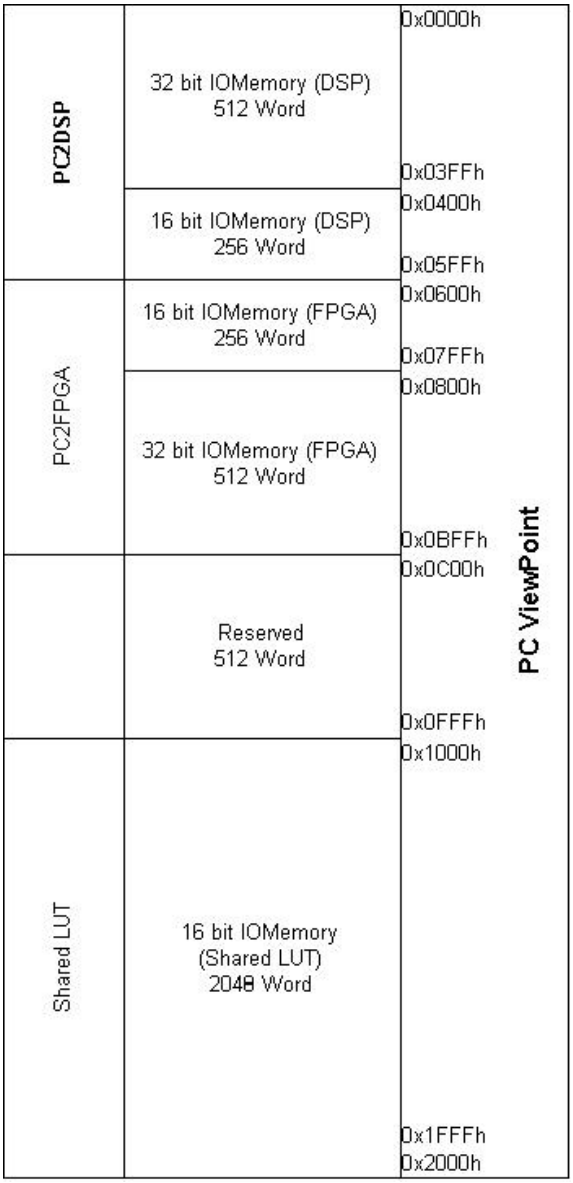



Figure 4.1. HRT Tandem Shared memory

for an available address block and allocates this memory; the start address reached is then written on a FPGA register mapped in the PORT I/O range which is fixed and known; from now PC and EKU can communicate using the MEMORY I/O range.

In order to realize a hard real time co-processing between DSP tasks and PC tasks a synchronization between them is required. The easily way to perform a task synchronization is to use the interrupts lines.
As mentioned is chapter 3 one of the system bus interrupts is dedicated to the PC-DSP communication. In order to trigger a DSP service routine the PC code writes a value into a dedicated HRT Tandem register (PORT I/O); the FPGA is sensible to the register writing (the value written does not matter) and asserts the relative SYS_INT line.
The synchronization in the other direction is similar: the DSP asserts a predefined bit of a system bus register; the FPGA recognizes the bit value and asserts the CPU hardware interrupt; the FPGA resets the bit written by the DSP in order to be ready for another interrupt request.
The CPU hardware interrupt line is a parameter of the EKU-ISA driver; this selection is written by the driver to a PORT I/O register at the start up in order to instruct the FPGA on the interrupt line to assert when needed.

### 4.1.2 Host Tandem

As seen in chapter 2 the link between the EKU and a host PC can be performed in two ways:

- Serial RS232/RS422 communication protocol
- ISA protocol

This connection allows to acquire real time variables, in order to plot data while the system is running, to tune the control parameters in real time and to monitor the time occupation of the tasks running on the DSP.
The serial connection is managed, by the EKU point of view, directly by the DSP and this solution is very cheap. The main limitations are the low throughput and the short distance between the target and the host PC.
The PC solution allows to transfer data between target and host in two different ways:

- FIFO (First In First Out) mode
- DMA (Direct Access Memory) mode

FIFO mode consists in two FIFOs, managed by the ISA-core IP, allowing data exchange in both PC to DSP direction (PC2DSP FIFO) and DSP to PC direction (DSP2PC FIFO).
PC2DSP FIFO is used by the PC to set DSP and FPGA control parameters and to send commands for variables acquisitions while DSP2PC FIFO is used by the DSP to send to the PC the acquired variables.
PC2DSP FIFO is written by the PC simply accessing a dedicated PORT I/O register; once that register is read, the PC has information about the number of words stored on the FIFO, useful to avoid writing on a full FIFO.
The other end-point of this FIFO is accessed by the DSP through a system bus register: with each read operation a new value is output from the FIFO and the number of elements present on it is decremented.
DSP2PC FIFO is filled by the DSP writing on a dedicated system bus register; DSP can know the number of words stored on the FIFO reading the same register.
The output of this FIFO is accessed by the PC when it reads another dedicated PORT I/O register.
FIFO mode is completely asynchronous and it does not use any interrupts.
DMA mode is only used in the DSP to PC direction for real time variables acquisition.
It consists in a direct access to the PC RAM: DSP fills a dedicated FIFO instead of the DSP2PC one, FPGA automatically requires the access to PC DMA controller and starts transferring data from the FIFO to the PC RAM. Data flow is divided into pages of 4096 words each; at the end of each page the FPGA triggers the PC using a dedicated interrupt line. Once the page is received, PC acquires data and acknowledge the data reception writing a dedicated PORT I/O register; FPGA can now start sending another page until all the variables are sent.
The interrupt line and the DMA channel number are EKU-ISA driver parameters which writes this values to a PORT I/O register at the start up in order to instruct the FPGA.
DMA mode can reach a throughput of about $2MByte/S$ and allows to acquire up to tens megabytes every time.

## 4.2 Communication Interface

The communication interface discussed here is related to networks and grids between electronic control units. It is divided into the standard protocols and a custom protocol; the first ones are used to interconnect the EKU with a standard network formed by different and various digital platforms. The second is intended to create an Hard Real Time Grid between EKUs for applications that need high computational power or for redundancy reasons.
All these communications pass through the FPGA because they all use the digital connector that is directly connected to the FPGA.

### 4.2.1 Standard Networks

As mentioned in the previous chapters, the standard networks, which the EKU is able to interconnect to, are, at the moment:

- CAN bus
- RS232, RS422, RS485
- All the protocol implemented in the Anybus products developed by HMS company

The FlexRay expansion is, at the moment, under investigation and development.
All these protocols are implemented in external devices mounted on communication boards and all these boards are connected to the same communication connector of the EKU. For this reason these boards are mutually exclusive.
The drivers for all these devices are implemented in the firmware operating system, but only one of these can be used at a time.
To do this a constant named `ComM` is provided; this constant can assume the values `CAN, ANYBUS and UART)` and the user has to set this constant accordingly with the board mounted on the stack.

```
...
type   ComM_select     is (CAN,ANYBUS,UART);
...
-- CAN enabled / ANYBUS disabled / UART disabled
constant ComM   : ComM_select := CAN;

-- ANYBUS enabled / CAN disabled / UART disabled
--constant ComM   : ComM_select := ANYBUS;
```

```
-- UART enabled / ANYBUS disabled / CAN disabled
--constant ComM   : ComM_select := UART;
...
```

The firmware operating system, based on the value set, instantiates the correct IP-core and links the communication connector pins to the IP-core input and output signals.
A change of this constant value requires the firmware to be recompiled because the EKU needs to power up with the correct firmware in order to avoid collision on the communication bus at the start up.
The total pins available for the external standard communication in the EKU board is 30; these pins are called COMMxx, where xx goes from 00 to 29, and are set as bidirectional pins in the top level entity.

```
entity top_level is
port{
      ...
      COMM00        : INOUT STD_LOGIC;
      COMM01        : INOUT STD_LOGIC;
      COMM02        : INOUT STD_LOGIC;
      ...
      COMM29        : INOUT STD_LOGIC;
      ...
      }
end top_level;
```

Two buses (COMM_I and COMM_O) are used to propagate the communication bus through the FPGA code. A pins direction bus is provided (COMM_DIR(xx) in order to instruct the FPGA output drivers as for the field bus pins.

```
begin
...
COMM00      <= COMM_O(0)  when COMM_DIR(0)  = '1' else 'Z';
COMM01      <= COMM_O(1)  when COMM_DIR(1)  = '1' else 'Z';
COMM02      <= COMM_O(2)  when COMM_DIR(2)  = '1' else 'Z';
...
COMM29      <= COMM_O(29) when COMM_DIR(29) = '1' else 'Z';
...
...
```

```
COMM_I(0)   <= COMM00;
COMM_I(1)   <= COMM01;
COMM_I(2)   <= COMM02;
...
COMM_I(29)  <= COMM29;
...
end;
```

The pins assignment related to the value assumed by the constant `ComM` is performed using the construct `with select`; this construct implements a combinational logic multiplexer and the assignment is performed in a time as short as possible.
An example of this assignment for some of the communication pins is presented below:

```
with ComM select
  COMM_O( 0)     <= can1_CSn       when CAN,
                    '1'            when ANYBUS,
                    uart_HDPLX( 0) when UART;
with ComM select
  COMM_DIR( 0)   <= OUTPUT         when CAN,
                    INPUT          when ANYBUS,
                    OUTPUT         when UART;

with ComM select
  COMM_O( 1)     <= can1_WRn       when CAN,
                    Dsp_WEn        when ANYBUS,
                    uart_SHDNn( 0) when UART;
with ComM select
  COMM_DIR( 1)   <= OUTPUT         when CAN,
                    OUTPUT         when ANYBUS,
                    OUTPUT         when UART;
```

### 4.2.2 HRT Grid

HRT Grid is, first of all, a concept: more than one EKU hardly connected in order to realize a computational grid with hard real time characteristics.
A grid is intended for few data exchange between components in scheduled period, in contrast with a network where large amount of data is exchanged as soon as possible.
The EKU composing the grid usually work on the same application and often on the same plant when the grid is used to implement a redundant system.
A smart way to work, when the application is common, is:

- EKUs share the same resources

- EKUs are synchronized

To satisfy these constraints a custom communication protocol has been developed. The grid topology chosen is shared parallel bus with dedicated data and address bus; few control signals are used to manage the arbitration.
In the actual implementation the maximum number of EKUs in the grid is limited to 8; data bus width is 16 bits (in order to be compliant with the system bus); the address bus is 5 bits wide; the maximum distance between EKUs depend on the flat cable used since the main issue is signal integrity.
The implementation has been performed completely in FGPA using the device pin drivers to access the bus while the DSP application code accesses the grid resources by means of the system bus. The ISO/OSI model is illustrated in figure 4.2: in the left side the environments (Soft Ware, Firm Ware and Conditioning Ware) where the layers are implemented are shown while in the right side the EKU target devices are linked.
The physical layer is a simple flat cable directly driven by the FPGA pins for the

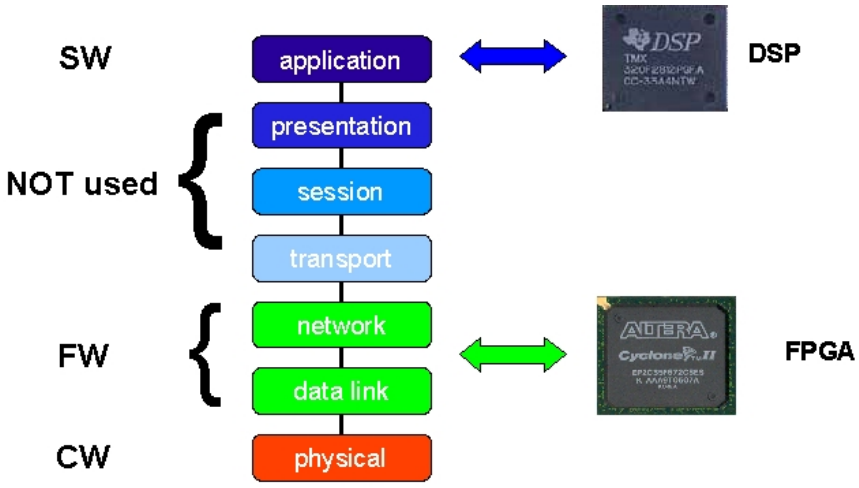



Figure 4.2. HRT Grid ISO/OSI model and target devices

address and data buses, while open drain configuration (with a pull-up resistor) for control signals is used.

### Shared memory

To perform the resources sharing the HRT grid has been developed basing on the shared memory concept. Since it is difficult to manage a de-localized memory, the solution that has been implemented is a distributed multi-port memory:

- Every EKU accesses to the shared variables from its own local memory (without accessing the bus)
- Variables update is performed accessing the HRT Grid bus
- Local memory is updated from the bus (even the memory located on the EKU that is performing the update operation)

From the system level point of view the shared memory has as many concurrent read ports as the number of EKUs in the grid and only one write port multiplexed between all the masters accessing the bus (see figure 4.3). With masters we mean the EKUs in the grid enabled to write on the bus.

The main issue regarding the use of a shared bus is the bus contention and then

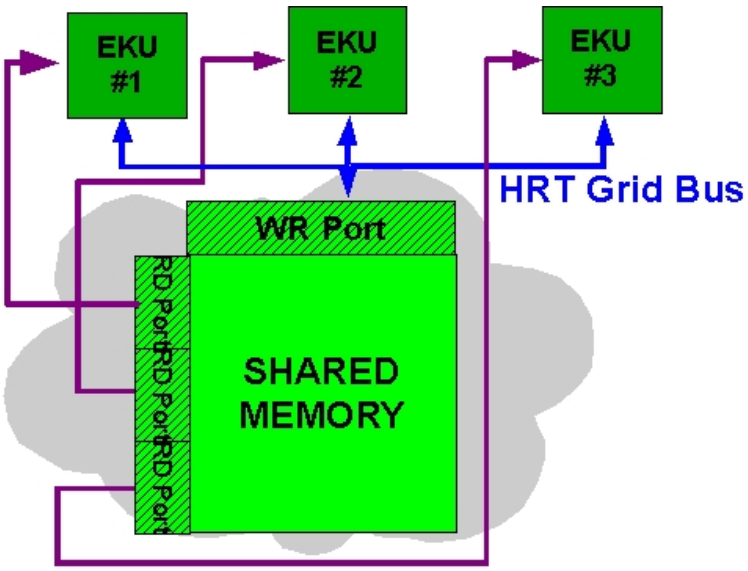



Figure 4.3. HRT Grid shared memory: Writes use the bus; Reads are executed locally

the protocol arbitration. To overcome this problem, a solution based on the token passing algorithm has been chosen (see figure 4.4). One of the master EKUs in the grid has the task of generating the token to pass between the other masters; this EKU will be called token generator.

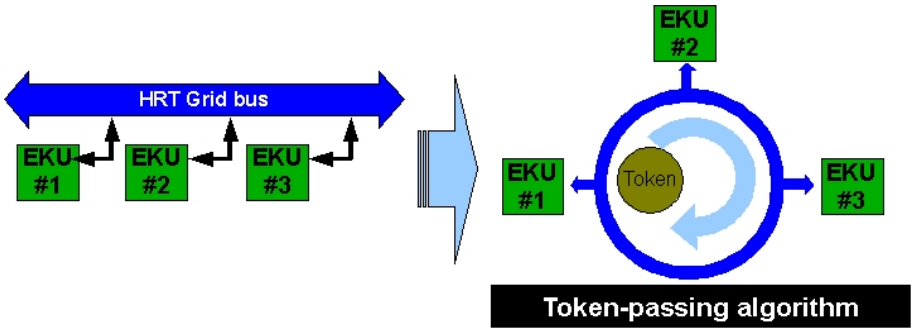



Figure 4.4. Data link implementation: Token Passing Algorithm

The token is constituted by a three bits number indicating the identification number (ID) of the master enabled to access the bus in that slot.
The complete operating flow concerning the token passing implementation is listed here:

- While the bus is in idle state, the token generator drives the three least significant address bits with the token number
- Until a master EKU access the bus, the token generator increments the token ID every $40nS$; the token IDs generated include only the master EKUs IDs
- When a master EKU has a word to write on shared memory, recognizes proper token ID and assert a control signal (busy, active low) to indicate a new bus transaction
- The token generator releases the bus and puts the previously driven address bits to high impedance
- when the transaction has finished the EKU master responsible of that deasserts the busy signal and the token generator starts again to generate the ID sequence

The token generating process is implemented in all the EKUs of the grid but it is enabled only on one of these; this process is concurrent with the others therefore the token generator acts on the bus as all the others EKUs (writes on the bus when the token ID is correct).
The write transaction is very simple:

- The master EKU which is accessing the bus drives the busy signal (as seen before), the address bus, the data bus and then a control signal (strobe) to indicate that the data on the bus are valid

- all the EKUs in the grid (included those that are not master and that is writing on the bus) acquire the data and write it on the correspondent location of the local memory
- when the data has been saved each EKU asserts a control signal (ACK, active high, wired-and) to indicate the end of the update operation
- when the master EKU recognize the ACL signal at high level release the bus de-asserting the strobe, data and address signals
- when the strobe signal is de-asserted, each EKU release the ACK signal and the token generator accesses the bus
- if, after a time-out time, the ACK signal is still inactive, a fault interrupt is triggered on the DSP to indicate an error in the grid

All the write process takes about $100nS$.
The total amount of shared memory is 32 words due to the 5 bits for the address bus; a location is reserved for the start up configuration of the grid; a second location is dedicated for the grid synchronization as we will see later in this section, while the other 30 words are divided into two parts:

- DSP shared memory (16 words)
- FPGA shared registers (14 words)

The DSP shared memory is dedicated to the real time variables of the codes running on the DSPs. Sometimes these variables are floating point or often in IQ format. IQ format is a Texas Instruments variation of the Q format to represent the numbers in fixed point mathematics; in the IQ format the data width is always 32 bits and only the number of bits after the point is indicated (i.e. IQ14 means $32 - 14 = 18$ bits for the signed integer field and 14 bits after the `decimal` point).
To write a 32 bits variable on the shared memory, two accesses must be performed and it means that a data inconsistency may occur (see figure 4.5).

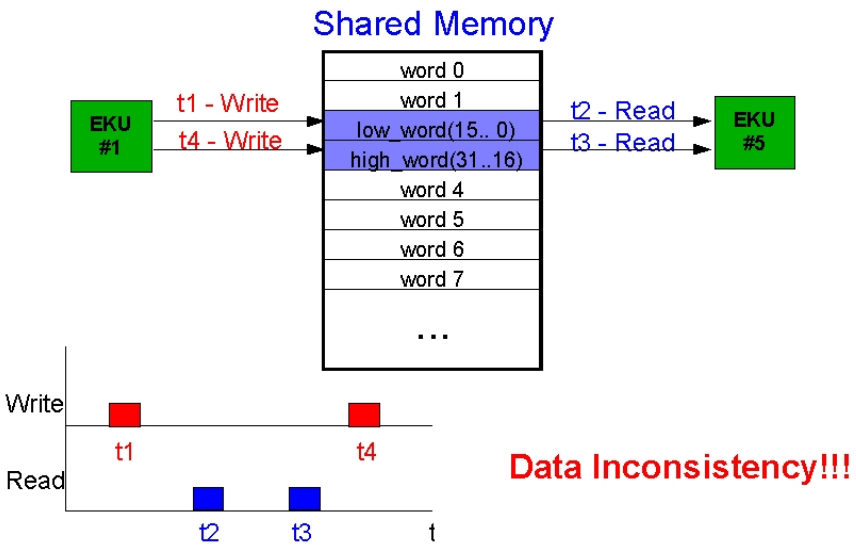



Figure 4.5. Example of 32 bits variable writing (EKU 1) and erroneous data reading (EKU 5)

To overcome this issue a particular protocol feature has been implemented: the DSP has the possibility to write up to four words in the same transaction (burst transfer) in order to avoid data inconsistency for two 32 bits variables or up to four 16 bits variables that must be updated simultaneously (for example the filter coefficients). To perform the burst transaction the DSP writes the number of words that must be transferred on a dedicated register of the HRT Grid config IP; when the token ID is equal to those of the EKU ID, the HRT Grid IP starts the transaction similarly to the single transaction but, at the end of the first word writing, the busy signal is not released and new address and data values are set. When the burst transaction is finished (the last word has been written), the busy signal is released and the bus come back in idle state.
Data inconsistency is avoided because the local memories update is performed when the busy signal come back high (busy signal is active low); An example with the temporal actions is illustrated in figure 4.6.

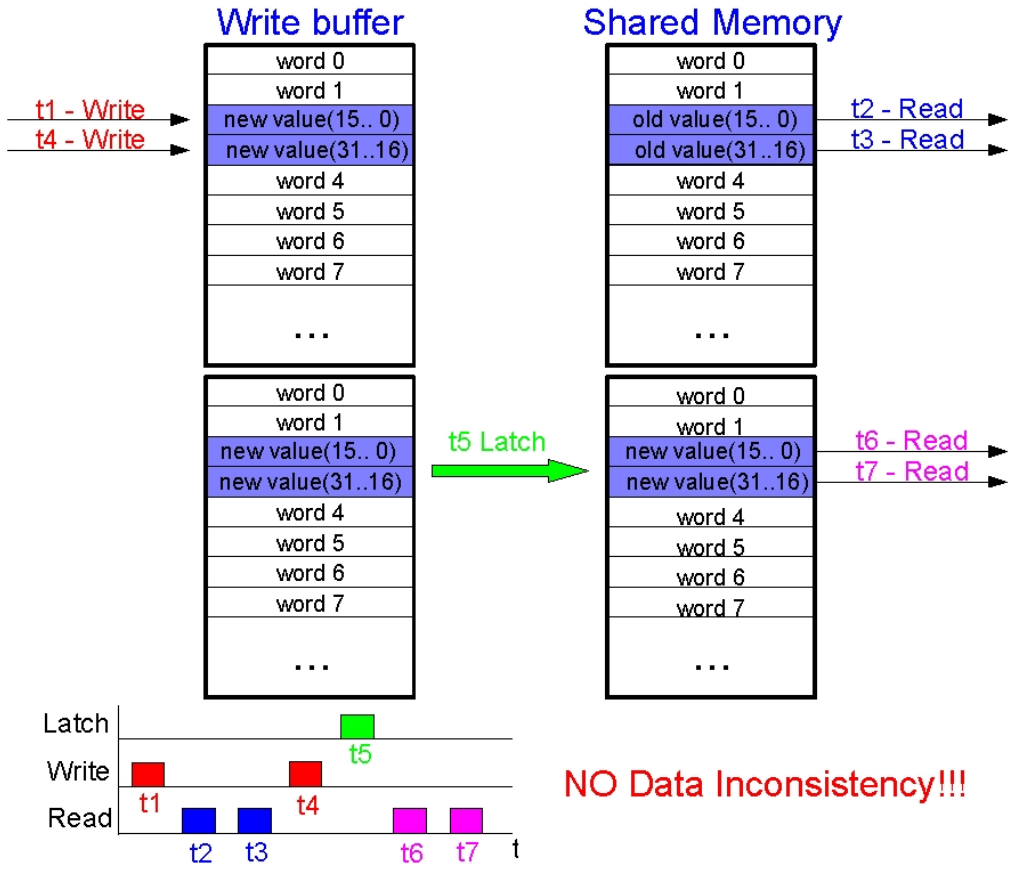



Figure 4.6. Temporal flow of a 32 bits variable updating in burst mode

To avoid any other data inconsistency problems, the EKU that is reading from the shared memory has the possibility to hold the bus until all the variables, of a consistent set, has been read from the local memory; in this way the latch operation, seen in figure 4.6, does not occur between two read operations.

The FPGA shared memory is dedicated to the registers managed by the firmware user IPs; these registers can be, for example, the output of a motor encoder interface that the developer wants to share between the EKUs in the grid in order to minimize the resource redundancy, or any other product of a firmware IP the user has implemented.
In the user template 14 data registers and a control register have been provided in order to manage the registers share on the grid. The control register acts as a trigger: each bit of this register (first 14 bits) is related to a particular register to share; when one of these bits is asserted, the relative register is sent to the HRT

Grid manager IP which waits the token and then writes the value on the bus.
The writing procedure is equal to that of the DSP shared memory but only single transactions are allowed.
The registers triggering can also be performed by the DSP of the same EKU: a register managed by the HRT Grid config IP has the same functionality of that of the user template; in this way an FPGA register is shared when a user IP asserts the correspondent bit in the user template OR the DSP asserts the correspondent bit by means of the system bus.
With this two possibilities the system scheduler can be located either on the FPGA or on the DSP where, usually, the main program is running.

### Distributed Scheduler

In order to satisfy the constraint about the EKU synchronization one of the 32 shared register is dedicated to the DSP interrupt service routines trigger.
As mentioned in chapter 3, three of the eight system interrupt manager lines are dedicated to the HRT Grid ISRs (ISR1, ISR2 and ISR3); a fourth interrupt service routine is reserved for faults notification.
As seen the maximum number of EKU in the grid is 8. The trigger register has been virtually divided into eight fields of two bits each (one field for each EKU); these two bits can assume four numerical values (00b, 01b, 10b, 11b). At each of these values is associated a triggering function:

- 00b does not produce effects (this is the idle situation)
- 01b produces the assertion of the ISR1
- 10b produces the assertion of the ISR2
- 11b produces the assertion of the ISR3

For example, writing 0x0020 on the trigger register will assert the ISR2 on the EKU number 3 in the grid, while writing 0xC821 on the same register will assert the ISR1 on the EKU number 1, the ISR2 on the EKUs number 3 and number six, and the ISR3 on the EKU number 8.
The trigger register is accessible by the DSP through a system bus register managed by the HRT Grid registers IP.
To make the triggering operation completely autonomous a particular IP called Scheduler has been implemented.
The task of the Scheduler IP is to trigger the ISRs on all the EKUs in the grid when particular events occur. An example can be the time driven scheduler where a counter is used as time base and when the IP recognize a particular time event, it

triggers the relative distributed ISRs.
The association between the events and the triggering patterns is stored in a two columns look up table of 16 lines each. A graphical example is given in figure 4.7.
Instead of the time domain, another domain often used is the angular one, especially

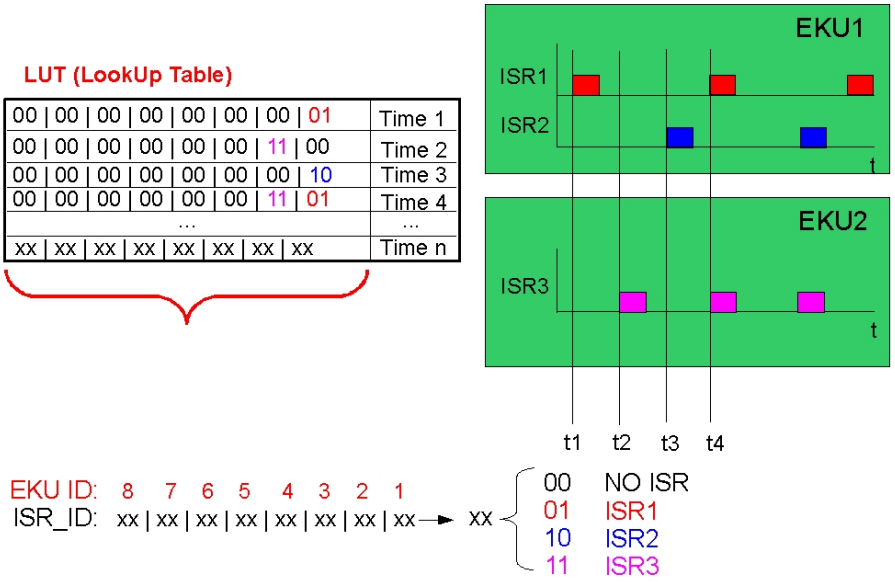



Figure 4.7. Time scheduling of three ISRs on two different EKUs in the grid

in the system where the HRT Grid is used to substitute the mechanical camshafts in industrial machines. In this case instead of the time, the Scheduler IP input is the angle (read for example by means of an encoder) and the LUT is filled with angular values and triggering patterns. A graphical example of the angular LUT filling is shown in figure 4.8.

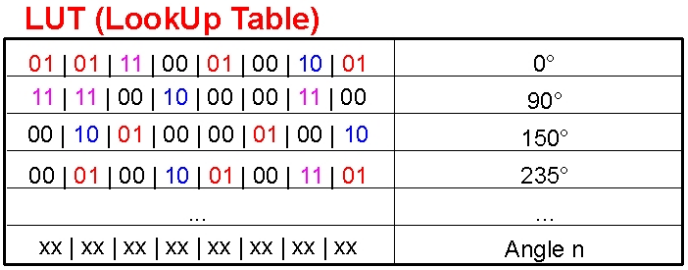



Figure 4.8. Scheduler IP LUT filled with angular values associated to triggering patterns

## 4.3 From/To Field

It is known that the FPGA is a exclusively digital device and for this reason it has not embedded analog peripherals.
Unfortunately (or not?) the real world is analogical and being able to interface an ECU to analogical quantities is mandatory.
The DSP has embedded 16 ADC channels 12 bits able to acquire at up to 1M samples per second. The values acquired can be shared with the FPGA code by means of the system bus.
The DSP has not, however, an embedded DAC peripheral; for this reason an external DAC device has been provided on the EKU and the management is delegated to the FPGA operating system by means of an IP-core that acts as driver.
Again, the inputs of the DAC device are shared with the DSP by means of the system bus.

### 4.3.1 Digital Field Bus

During the EKU development the field bus specification were written in order to standardize the field interface for further EKU version and for field module development.
From the EKU point of view the field bus is divided into two distinct connectors:

- Digital Field Bus connector
- Motor/analog connector

The Digital Field Bus connector carries, in addition to power supply and ground signals, 68 digital I/O pins directly managed by the FPGA.
Even if the FPGA can handle different electrical standards, in this version the standard used is the LVTTL (Low Voltage TTL) which means single ended $3.3V$ level.
Motor/analog connector carries both digital and analog signals; the firsts and the analog input signals are directly managed by the DSP while the analog output signals are fed by the external DAC device driven by the FPGA.
The FPGA pin drivers have the possibility to be set as output, input or high impedance; in any case the high impedance condition is equivalent to the input one and, moreover, while a pin is set as output the read operation is always enabled. In short, setting an FPGA pin as bidirectional (INOUT) the read can always be performed.
Taking advantage of this property, in the FPGA operating system the field bus pins (FB_PIOxx) are set as bidirectional in the top level entity and the field bus is then propagated through the FPGA project separated into two buses: field bus input bus

(PIO_I(xx)) and field bus output bus (PIO_O(xx)).
The input bus is always fed by the field bus pins while the output bus drives the field bus pins only when the direction for that pin is set as output (PIN_DIR(xx)).

```
entity top_level is
port{
      ...
      FB_PIO00      : INOUT   STD_LOGIC;
      FB_PIO01      : INOUT   STD_LOGIC;
      ...
      }
end top_level;
...
...
begin
...
FB_PIO00    <= PIO_O(0)   when FB_DIR( 0) = OUTPUT else 'Z';
FB_PIO01    <= PIO_O(1)   when FB_DIR( 1) = OUTPUT else 'Z';
...
...
PIO_I(0)    <= FB_PIO00;
PIO_I(1)    <= FB_PIO01;
...
end;
```

The bus PIN_DIR is a custom type that can assume the values INPUT or OUTPUT. Then two buses for input/output signals and a bus that allows to set the direction for each pin of the field bus have been provided in the firmware user template.
Below an example is given. Two digital input signals (PIO_I(0) and PIO_I(2)) have to be acquired and saved on internal signals called example_in1 and example_in2; an internal signal example_out has to be output to the field bus pin PIO_O(1).

```
entity user_template is
port{
      ...
      PIO_I         : in  std_logic_vector(67 downto 0);
      PIO_O         : out std_logic_vector(67 downto 0);
      FB_PIO00      : out PIN_DIR;
      FB_PIO01      : out PIN_DIR;
      FB_PIO02      : out PIN_DIR;
```

```
      ...
      }
end user_template;

architecture struct of user_template is
...
signal example_in1  : std_logic;
signal example_in2  : std_logic;
signal example_out  : std_logic;
...
begin
...
example_in1 <= PIO_I(0);
example_in2 <= PIO_I(2);
...
PIO_O(0)    <= VCC;             FB_PIO00  <= INPUT;
PIO_O(1)    <= example_out;     FB_PIO01  <= OUTPUT;
PIO_O(2)    <= VCC;             FB_PIO02  <= INPUT;
...
end;
```

### 4.3.2 DAC Management

On the digital platform used to develop the firmware operating system has been provided a 8 channels 12 bits D/A converter device able to output each channel every $1\mu S$.
The eight outputs are independent but can be synchronized if required by the application.
The device communicates with the `host` via the SPI (Serial Parallel Interface) protocol. For this reason the main task of the DAC management IP-core is to perform a bridge between the wishbone protocol and the SPI protocol.
The SPI communication has been implemented in a single state machine process triggered by the wishbone state machine. The SPI clock speed is $25MHz$ and the data to be sent is 16 bits wide, so the time required for a single transmission is about $640nS$; using all 8 channels together, with the clock frequency chosen, the overall time needed to output the analog signals is about $10\mu S$.
First operation performed by the IP-core after the start up is the initialization of the DAC device:

- DAC device is powered up
- modalities of conversion are set

- number of used channels is set
- digital inputs are written with the value set in the operating system
- the conversion is triggered

During normal operation the master (DSP code through system bus or internal FPGA code) can access the eight digital input data as a register. After all the required data have been set, the conversion can be triggered to output synchronously with all the analog signals.
It is possible to instruct the device to convert the digital data inputs as soon as they are updated. In this way the output signals are asynchronous.

# Chapter 5

# Performances Evaluation

This chapter briefly shows the performances of the main parts of the EKU digital platform and the FPGA operating system.
These data can be useful to the developer who wants to decide how to divide his own application between the system components. Since the digital platform and its operating system allows to distribute the computation on three kind of devices: DSP, FPGA and PC (or micro processor).
The computational evaluations are executed exclusively on DSP and FPGA because many PCs can be used with the EKU and the performances (substantially) depend on the powerful and on the clock speed of the CPU. The floating point on the DSP is emulated since it has not a hardware floating point unit. The performances are evaluated on typical operations and computation used in control and mechatronic applications.

## 5.1 EKU Kernel Performances

The intention of this section is to give some information about the performances of the EKU kernel key components. The performances are expressed in terms of time consumption of each single operation. The tests on the DSP are performed with the code running on internal DSP RAM and the main DSP clock at $150MHz$. The test on the FPGA are performed with a system clock at $100MHz$ since with the current FPGA device speed grade the clock frequency cannot reach the $150MHz$ without particular tricks hardly to be implemented.
The operations used for the performances evaluation are:

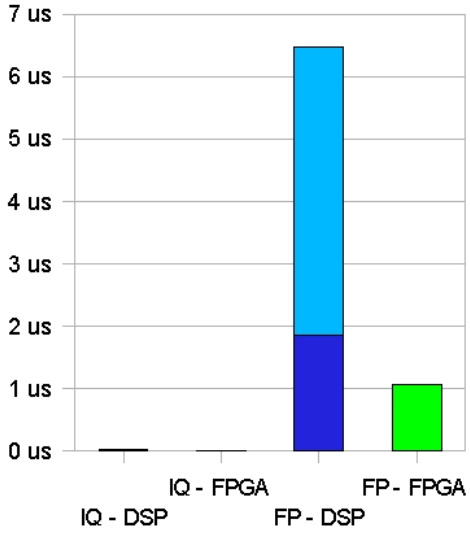
- Fixed and Floating point Addition/Subtraction
- Fixed and Floating point Multiplication

- Fixed and Floating point Division
- fixed point to floating point conversion (IQ to FP)
- floating point to fixed point conversion (FP to IQ)

For the floating point operations executed into the FPGA, also the time spent to transfer the operands and the result between the DSP and the FPGA has been taken into account since the main objective of this performance evaluation is to give the complete information in order to perform a efficient DSP-FPGA co-processing.
The time required by the system bus to transfer a single word of 16 bits is exactly $160nS$. Therefore, the time taken by the system to transfer a floating point operand is $320nS$; a complete floating point operation executed in DSP-FPGA co-processing takes $320ns + 320nS + 320nS = 960nS$ to transfer the two operands and the result.

### Addition/Subtraction

The addition and subtraction operations take the same execution time and for this reason they are treated together. The Fixed point computation takes $20nS$ on the DSP and $10nS$ on the FPGA, while the floating point computation time is in the range $1.86\mu S \div 6.48\mu S$ on the DSP and it takes $100nS$ on the FPGA ($1.06\mu S$ including the system bus time). The time spent by the floating point operations on the DSP depends on the values of the operand and precisely on how far are the operand values.
A summary table and the graph are given below:



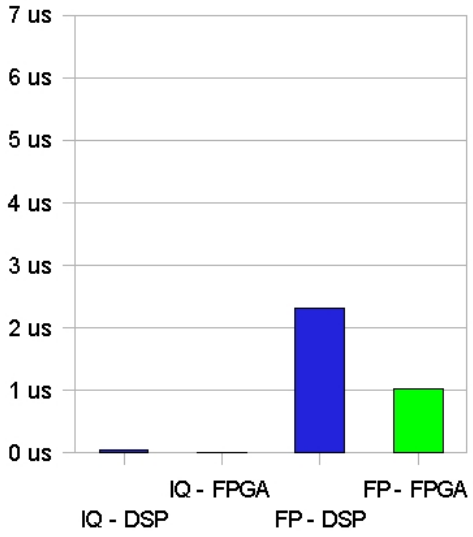

| | Time (us) |
|---|---|
| IQ – DSP | 0.020 |
| IQ – FPGA | 0.010 |
| FP – DSP | |
| Min* | 1.860 |
| Max* | 6.480 |
| *1° operand = 0* | *0.710* |
| *2° operand = 0* | *1.450* |
| FP - FPGA | 1.060 |
| *Execution only* | *0.100* |

** approximative time*

The floating point FPGA implementation uses 2832 logic elements.

### Multiplication

The fixed point multiplication computation takes $50nS$ on the DSP and $10nS$ on the FPGA, while the floating point computation takes $2.307\mu S$ on the DSP and $60nS$ on the FPGA ($1.02\mu S$ including the system bus time).
A summary table and the graph are given below:

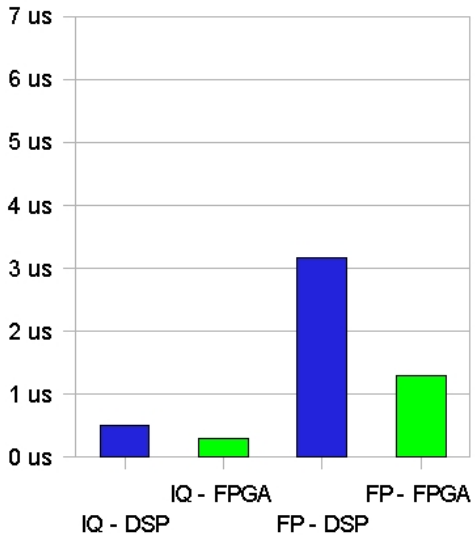



| | Time (us) |
|---|---|
| IQ – DSP | 0.050 |
| IQ – FPGA | 0.010 |
| FP – DSP | |
| Total | 2.307 |
| *1° operand = 0* | *0.547* |
| *2° operand = 0* | *0.813* |
| FP – FPGA | 1.020 |
| *Exec only* | *0.060* |

The floating point FPGA implementation uses only 849 logic elements since the FPGA uses the embedded hardware multipliers to perform the operation.

### Division

The fixed point division operation takes $510nS$ on the DSP and $300nS$ on the FPGA, while the floating point one takes $3.16\mu S$ on the DSP and $340nS$ on the FPGA ($1.3\mu S$ including the system bus time).
A summary table and the graph are given below:

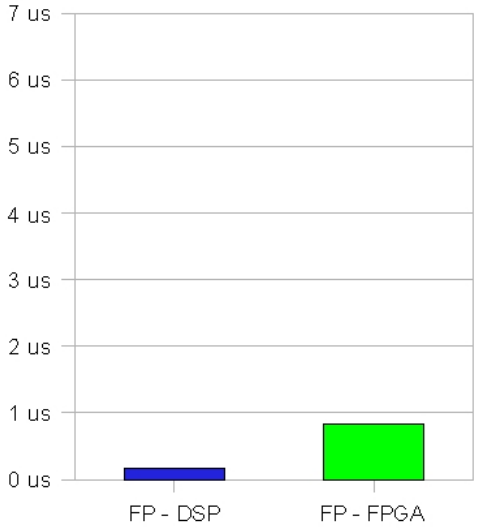



| | Time (us) |
|---|---|
| IQ – DSP | 0.510 |
| IQ – FPGA | 0.300 |
| FP – DSP | |
| Total | 3.160 |
| *1° operand = 0* | *0.570* |
| *2° operand = 0* | *1.270* |
| FP – FPGA | 1.300 |
| *Exec only* | *0.340* |

The floating point FPGA implementation takes up 10000 logic elements since, in order to keep the clock at $100MHz$, many pipeline stages are required.

### IQ to FP

The fixed point to floating point conversion is necessary when a data from the field (ADC channel or a digital input sensor) must be included into a floating point algorithm.
This conversion takes $174nS$ on the DSP and $40nS$ on the FPGA ($840nS$ including the system bus time). In this case the operands to be exchanged on the system bus are three: the input (32 bits), the output (32 bits) and the number of `decimal` bits (16 bits).
A summary table and the graph are given below:

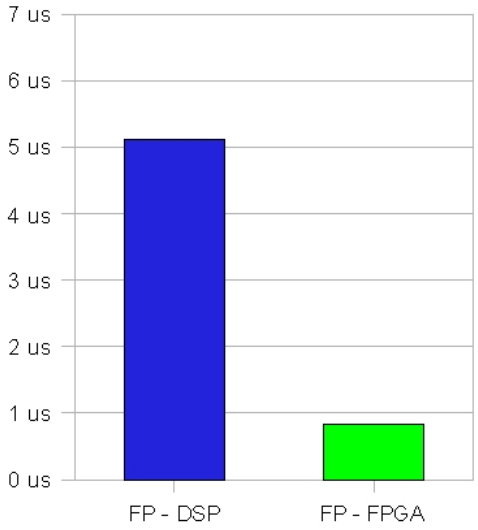



| | Time (us) |
|---|---|
| **FP – DSP** | **0.174** |
| **FP – FPGA** | **0.840** |
| *Exec only* | *0.040* |

The FPGA implementation of the conversion fills up 1105 logic elements.

### FP to IQ

The floating point to fixed point conversion is necessary when a result of a floating point algorithm must be sent to the field in digital form or in analogical form by means of a DAC device.
This conversion takes $5.125\mu S$ on the DSP and $40nS$ on the FPGA ($840nS$ including the system bus time). Even in this case the operands exchanged on the system bus are three: two 32 bits variables and one 16 bits variable.
A summary table and the graph are given below:




| | Time (us) |
|---|---|
| **FP – DSP** | **5.125** |
| *operand = 0* | *2.511* |
| **FP – FPGA** | **0.840** |
| *Exec only* | *0.040* |

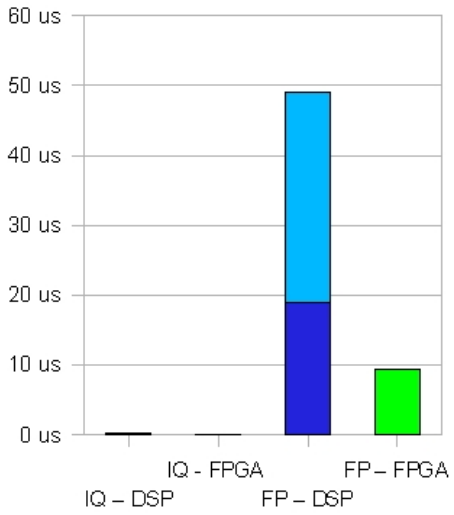
The FPGA implementation of the conversion takes up 520 logic elements.

### Second Order Section Filter

One of the most used filter in mechatronics is the so called second order section filter. Its discretized form is:

$$y_i = A_1 y_{i-1} + A_2 y_{i-2} + B_0 x_i + B_1 x_{i-1} + B_2 x_{i-2} \tag{5.1}$$

The fixed point computation of this filter takes $310nS$ on the DSP and $100nS$ on the FPGA, while the floating point one is in the range $18.98\mu S \div 30\mu S$ on the DSP and it takes $700nS$ on the FPGA ($9.34\mu S$ including the system bus time).
A summary table and the graph are given below:




| | Time (us) |
|---|---|
| IQ – DSP | 0.310 |
| IQ –FPGA | 0.100 |
| FP – DSP | |
| Min* | 18.980 |
| Max* | 30.000 |
| FP - FPGA | 9.340 |
| *Exec only* | *0.700* |

** approximative time*

### PID with anti-windup

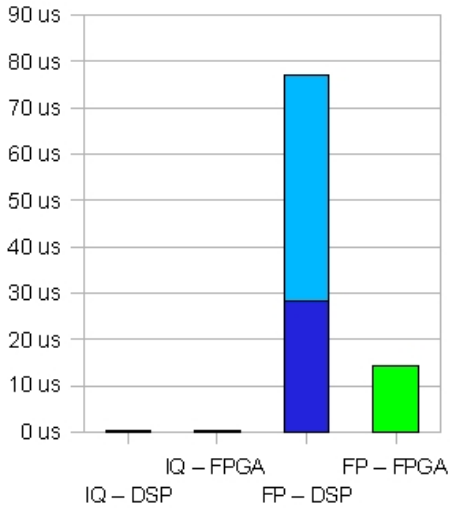
The most used controller in mechatronics is the PID (Proportional Integral Derivative) controller; the computation has been evaluated for the complete controller including the anti-windup branch. The mathematical form of the controller is:

$$u_i = k_c \left(P_i + D_i + I_i\right) \tag{5.2}$$

where:

$$\begin{cases} P_i = e_i \\ D_i = C_1 (e_i - e_{i-1}) + C_2 D_{i-1} \\ I_i = k_i (e_i + e_{i-1}) + I_{i-1} + k_a (u_{i-1} - u_{Pi-1}) \end{cases} \tag{5.3}$$

$C_1$ and $C_2$ depend on $k_d$ (derivative gain) and $N_d$ (term for the pole at high frequency). With $u_P$ is intended the pre-saturation output command.
The fixed point computation of this controller takes $430nS$ on the DSP and $150nS$ on the FPGA, while the floating point computation is in the range $28.39\mu S \div 48.55\mu S$ on the DSP while it takes $1.1\mu S$ on the FPGA ($14.4\mu S$ including the system bus time).
A summary table and the graph are given below:




| | Time (us) |
|---|---|
| IQ – DSP | 0.430 |
| IQ – FPGA | 0.150 |
| FP – DSP | |
| Min* | 28.390 |
| Max* | 48.550 |
| FP - FPGA | 14.400 |
| *Exec only* | *1.100* |

** approximative time*

## 5.2 Tandem and Grid Performances

### HRT and Host Tandem

As explained in section 4.1 the HRT Tandem is the use of the ISA protocol to realize a co-processing system between the EKU and the PC.
To perform a hard real time system, it is necessary that a hard real time operating system runs on the PC; in our tests the PC operating system is a Linux distribution with the RTAI extension.
The Host Tandem is intended to realize a host target link and it does not require a real time operating system running on the PC. The communication regarding the host Tandem is performed both in standard mode (PC acts as ISA master) and in

DMA mode (EKU is the master of the bus).
As mentioned before the operation performances depend exclusively on the PC CPU power and for this reason no performance evaluations has been performed. The most useful data that can be evaluated is the time taken by the communication operations.
Both standard and DMA operations take $1\mu S$ for each 16 bits data exchange. The total throughput for the DMA mode is then $2MByte/S$.
By the HRT Tandem point of view another value to be evaluated is the PC interrupt recognition time which is $1\mu S$ with the Linux RTAI kernel and the PC-104 used in our tests.

### HRT Grid

As mentioned in section 4.2 the HRT Grid data link layer is based on the token passing algorithm. The token changes its value every $40nS$ in order to give enough time to the addressed EKU to recognize the token and hold the bus.
Every single transaction (included the trigger transaction) takes $100nS$ while the burst transaction takes $360nS$ for a four words packet.
The maximum time ($T_{max}$) a EKU has to wait before having the possibility to access the bus is

$$T_{max} = 7T_t + 7T_P = 7 \cdot 40nS + 7 \cdot 360nS = 2.8\mu S \tag{5.4}$$

where:

- 7 is the maximum number of EKU in the write queue (max EKUs in the grid−1)
- $T_t$ is the maximum time during which the token hold its value
- $T_P$ is the time spent to transfer a packet of 4 words (burst transaction)

With a period of $2.8\mu S$ it is possible to close a distributed control loop at a rate above $350KHz$.

# Part II

# Real Application: Driving piezoelectric stack actuators

The aim of the project is to develop an electronic module that integrates logic and power devices to drive piezoelectric stack actuators and demonstrate experimentally the results in terms of control of piezoelectric stack tip displacement on a test bench.
The electronic module controls up to four piezoelectric stack actuators and guarantees that the correct tip displacement is reached starting from a desired profile.
The various opening/closing phases of the actuators are tuned in terms of slew rate, timings and values to reach during all the controlled phase.
The control parameters are passed to the control unit by means of a host human machine interface or by an external electronic control unit that acts as a supervisor.
This part will illustrate all the passages of the design starting from the constitutive equations of the piezoelectric material up to the final architecture of the control law and implementation passing through:

- creation of a FEM model of the piezoelectric stack;
- construction of the modal residues model;
- FEM model validation;
- identification of the electrical equivalent circuit of the piezoelectric stack;
- design of the power driver circuit;
- design of the control loops;

A complete model validation is then performed and experimental results are presented.

# Chapter 6

# Context

## 6.1 Piezoelectric principle

The theory of the piezoelectric materials can be seen in [10], while a very good introduction on the piezoelectric principle, and on the piezoelectric driving strategies can be found in [12].
The main property of piezoelectric materials is to deform themselves when an electric signal is applied, and reciprocally concentrate charge on their surface when a mechanical stress is applied on.
Therefore, if correctly polarized, these materials can produce a force; unfortunately however, even if this force can be very strong, it is associated to a very short movement. To solve this problem an appropriate kinematism must be mounted between the piezoelectric stack and the object to move, in order to amplify the displacement produced by the stack.
The piezoelectric stack itself is not a compact block of material, but it is built by several layers one upon the other; inferior and superior surfaces of each element are delimited by a metallic thin plate, to which polarization voltage is applied. The upper layer of the stack is connected to the housing, while the bottom layer moves the pin that acts on the input port of the external kinematism, if present.
All the piezoelectric layers are electrically connected in parallel, so they are all polarized in the same direction of the electric field applied: the global mechanical strain is the sum of the strain of each element of the stack.
The piezoelectric actuator is constituted by a piezoelectric stack and an external housing including a `preloading` system for the piezoelectric stack.
In linear piezoelectricity the equations of linear elasticity are coupled to the charge equation of electrostatics by means of the piezoelectric constants. The aim of the following section is to obtain an analytical model of the piezoelectric stack and its casing in order to study the displacement of the free end of the stack and the system

dynamics.

### 6.1.1 Symbols and Units

Because of the anisotropic nature of PZT ceramics, piezoelectric effects depend on the direction. The polarization direction is established during the poling process by a strong electrical field applied between two electrodes.
Piezoelectric materials are characterized by several coefficients:

- $c_{ij}$: material compliance $[N/m^2]$
- $d_{ij}$: material impermittivity $[m/V]$
- $B_{ij}$: material deformability $[m/F]$
- $H_{ij}$: charge constant $[V/m]$
- $S_{ij}$: piezoelectric constant $[m^2/N]$

### 6.1.2 Hysteresis of the material

Open loop piezoelectric actuators exhibit hysteresis in their dielectric and electromagnetic large signal behavior. Hysteresis is based on crystalline polarization effects and molecular effects within the piezoelectric material.
The amount of hysteresis increases with increasing voltage applied to the actuator. The gap in the voltage/displacement curve typically goes from about 2% to a maximum of about $10\% \div 15\%$ under large signal conditions.
Hysteresis is observable in open loop operation; it can be reduced by charge control and virtually eliminated by closed loop operation (figure 6.1).

## 6.2 State of the art

Nowadays the piezoelectric actuators are used in many applications:

- Positioning of mirrors or lenses
- Alignment or deformation of fibers
- Positioning of tools
- Pick and Place machines
- Proportional valves

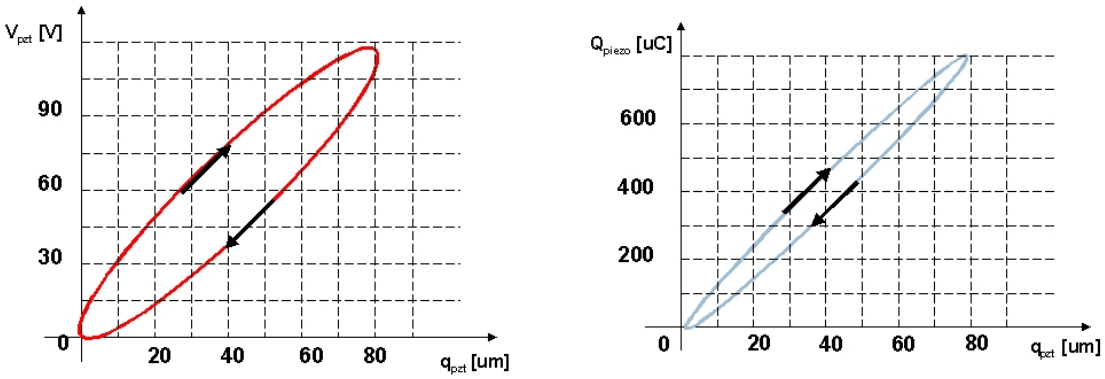



Figure 6.1. Theoric Hysteresis with voltage driven (left) and charge driven (right)

- Non-magnetic actuation
- and many others

The piezoelectric stack is an extraordinary linear actuator, capable of positioning resolution of nanometers and a stroke of hundreds micrometers.
On the market, two main companies that produce piezoelectric actuators and controllers can be found:

- Physik Instrumente (PI) [18] (fig. 6.2 and fig. 6.3)
- Thorlabs [20] (fig. 6.4)

Figure 6.2. Piezoelectric actuators picture taken from the PI web site


Figure 6.3. Others piezoelectric stacks taken from the PI web site


Figure 6.4. Piezoelectric products picture taken from the Thorlabs web site

Physik Instrumente and Thorlabs companies offer some controllers able to drive their actuators. These controllers are oriented to the nano and pico positioning. The output power is usually low and they are often based on linear amplifiers, that means low efficiency.
In some applications (like our one) the actuator has to act a high force in few microseconds and the efficiency of the system is a constraint; in these cases high power is needed and the linear amplifiers cannot be used because of their high losses. For these reason a switching power driver has been developed in this project. The literature on the switching drivers is very wide; some examples can be seen in [6], [19], [14], [13] and [1].

In our application it cannot be used a position sensor; a position open loop control strategy must be implemented.
There are two main different piezoelectric driving methods:

- Charge driving
- Voltage driving

Piezoelectric stack displacement is strictly related to the charge stored on it, so if a charge feedback amplifier is used, it is possible to correctly drive the actuator; the inputs are the charge and the external force (if present) ,while the outputs are the tip displacement and the voltage. As previously mentioned, using charge driving method the hysteresis effect is less than that obtainable using a voltage driving method. On the piezoelectric driving strategy, many arguments can be found in literature: [22], [8], [15], [5] and others.

# Chapter 7

# Model

## 7.1 FEM Model

For piezoelectric stack, we can create a FEM model, which is useful to study the displacements of the free end of the stack and the system dynamics.
This model in a first stage does not consider damping. This effect is introduced in a second stage taking the experimental response in resonance conditions into account. Including the mechanical dynamic behavior, this equivalent model can be easily implemented in electronic simulation software.

### 7.1.1 FEM analysis: main steps

The main steps of the construction of the piezoelectric stack and housing model are the following:

- STEP 1. Determination of the behavioral equations of a piezoelectric stack by means of FEM analysis, starting from constitutive equations of the material itself. Then we proceed to parameters identification basing on data provided
- STEP 2. Determination of mass , stiffness and coupling matrices for the entire stack, including housing
- STEP 3. Expression of the global equations into modal coordinates
- STEP 4. Modal reduction to the five most important modes of the system
- STEP 5. Conversion into state space domain obtaining an ABCD model or an equivalent transfer function model, that can be easily exploited

The consistency between the models obtained at various steps is verified at various levels e.g. comparing the transfer functions for different size of the model.

**Step 1: Constitutive equations of a piezoelectric element**

For a piezoelectric material, the equations we need are clearly standardized in the document ANSI/IEEE Std 176-1987, that expresses piezo equations and physical constants in SI units. The constitutive equations of a piezoelectric material at a microscopical level are the following:

$$\begin{cases} S = s^E T + dE \\ D = dT + \epsilon^T E \end{cases} \tag{7.1}$$

Or in an equivalent form:

$$\begin{cases} T = c^D S - hD \\ E = -hS + \beta D \end{cases} \tag{7.2}$$

The meaning of each symbol conventionally used in previous expressions is the following:

- $S$ strain of the material
- $s^E$ Young's modulus $[m^2/N]$
- $T$ mechanical stress $[N/m^2]$
- $d$ charge constant $[C/N]$
- $E$ electric field applied to the material $[V/m]$
- $D$ electric displacement $[C/m^2]$
- $\epsilon^T$ permittivity $[F/m]$
- $c^D$ elastic stiffness constant $[N/m^2]$
- $h$ piezoelectric constant $[V/m]$=$[N/C]$
- $\beta$ impermittivity constant $[m/F]$

Starting from the material constitutive equations 7.1 and 7.2 and using a FEM approach, it is possible to determine the equations that regulate the dynamic behavior of an entire piezoelectric stack.

The core of the actuator is a stack of piezoelectric layers. Each layer has a couple of electrodes to which the voltage is applied. Due to voltage application, an electric field is produced in the material, which is subject to mechanical strain. The entire stack was divided into nine elements; each element is constituted by a group of stacked layers.
From an electrical point of view, the layers are in parallel, while from a mechanical point of view they are in series. So, considering that the voltage is applied to the electrical nodes (or interfaces), it causes a deformation of the element. This deformation is described in terms of axial displacement of the two nodes located at the element extremities (figure 7.1).

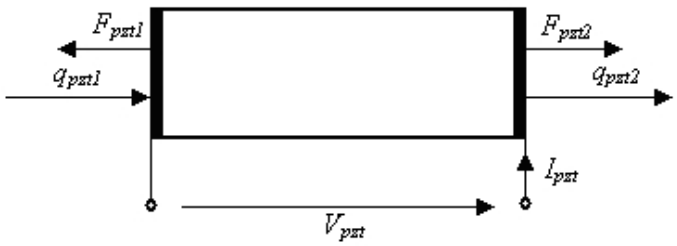



Figure 7.1. Forces and displacements at the extremes of an piezoelectric element

The electromechanical equations that regulate the behavior of the element are obtained from the expression of the potential and kinetic energy.
If we consider charge driving, the equations of the stack are:

$$\begin{cases} \underline{\underline{M}}\,\underline{\ddot{q}}_{stack}(t) + \underline{\underline{K}}^{Q}\,\underline{q}_{stack}(t) - \underline{\Gamma} Q_{piezo}(t) = \underline{F}_{ext}(t) \\ \underline{\Gamma}^{T}\,\underline{q}_{stack}(t) + \dfrac{1}{C_{pzt}} Q_{piezo}(t) = V_{pzt}(t) \end{cases} \tag{7.3}$$

While using voltage driving the equations are:

$$\begin{cases} \underline{\underline{M}}\,\underline{\ddot{q}}_{stack}(t) + \underline{\underline{K}}^{V}\,\underline{q}_{stack}(t) - \underline{\Theta} V_{pzt}(t) = \underline{F}_{ext}(t) \\ \underline{\Theta}^{T}\,\underline{q}_{stack}(t) + C_{pzt} V_{pzt}(t) = Q_{piezo}(t) \end{cases} \tag{7.4}$$

### Step 2: Determination of global matrices for entire stack

We are interested in the behavior of the piezoelectric actuator which is composed of piezoelectric stack and its case structure.
Stack case is a structure having mass and stiffness properties, which have interactions with the behavior of the stack itself. For model completeness, the influence of the stack case must also be considered.
In order to model the case and analyze its effects, we can discretize the structure into parts and then perform a FEM analysis. Furthermore, we know that the case is made of `invar`, so its density and Young modulus are reported in literature. The case is then modeled by meshing the structure using standard beam elements, whose sub-matrices are then appropriately assembled (figure 7.2).

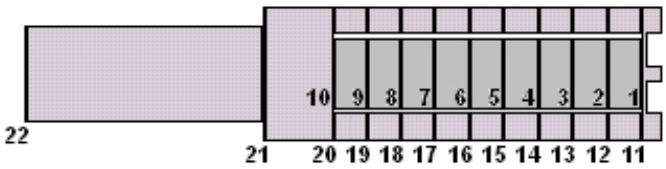



Figure 7.2. Standard enumerated beam elements of a piezoelectric stack

### Step 3: Conversion into modal coordinates

Then, once the global matrices of the piezoelectric stack and case are built, we can perform a modal analysis.
We start from the equations 7.4 in nodal coordinates and we express them in modal coordinates:

$$\begin{cases} \underline{\underline{m}}\ddot{\underline{\zeta}} + \underline{\underline{k}}\underline{\zeta} - \underline{\theta} V_{pzt} = \underline{f} \\ C V_{pzt} + \underline{\theta}^T \underline{\zeta} = -Q \end{cases} \tag{7.5}$$

This gives the advantage that each mode can be considered as decoupled from the others, making easier our mathematical work. In fact all the modal matrices are diagonal, so it is very easy to separate each single mode of the system.

**Step 4: Modal reduction to the five most important modes of the system**

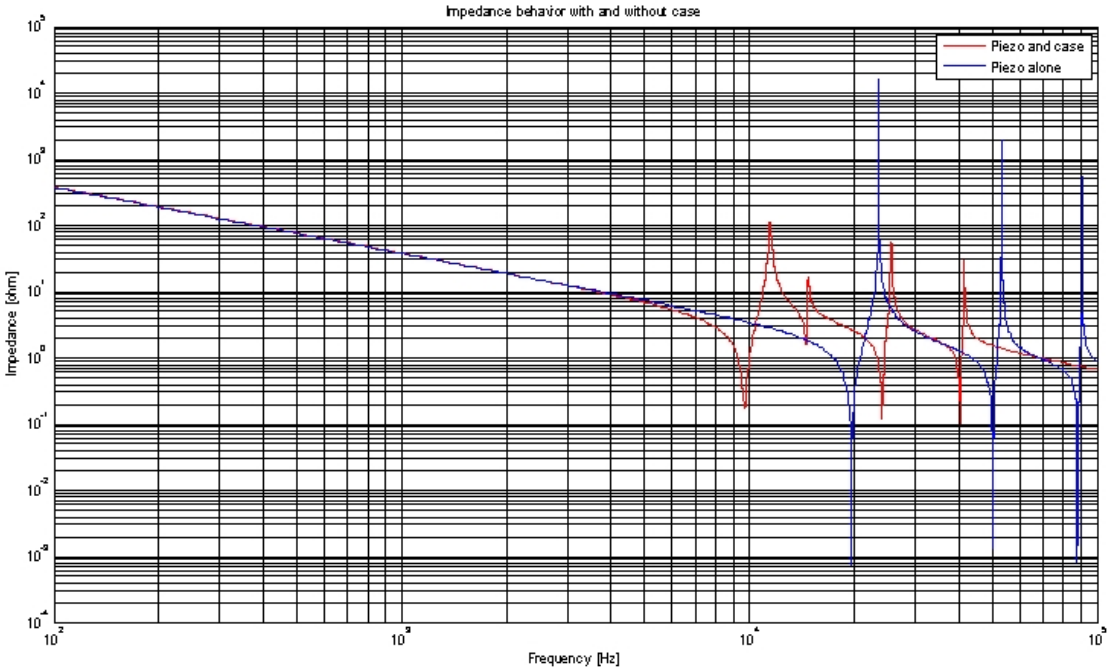
At this point, we can determine eigenvalues and eigenvectors of the system. The eigenvalues and eigenvectors are then sorted from low to high frequencies. The five most influent eigenvalues are responsible of the lowest five natural frequencies. The study is limited to these resonances because the frequency working range of the piezoelectric stack is lower than $10 - 15KHz$.
A Guyan method in performed to reduce the model. This method has been chosen because it guarantees that the low frequencies gain are kept.The model obtained from the reduction is formally equivalent to the initial model, but limited to the five lower natural frequencies. Therefore, the model is composed with 5-by-5 matrices.

**Step 5: Transformation into frequency domain and result plot**

At this point, a frequency plot can be obtained. In order to do this, we can convert the modal equations from time domain to frequency domain, using Laplace transformation:

$$\begin{cases} \underline{\underline{m}} s^2 \underline{\zeta} + \underline{\underline{k}} \underline{\zeta} - \underline{\theta} V_{pzt} = \underline{f} \\ \\ sCV_{pzt} + s\underline{\theta}^T \underline{\zeta} = -I \end{cases} \tag{7.6}$$

Then, we can describe the model in state space domain obtaining an ABCD model, giving as an input the current given to the piezoelectric stack and acquiring as an output the voltage at piezoelectric element connections. This expression is exactly equivalent to the transfer function form, but more compact and so preferred.
Impedance obtained by FEM analysis of stack and case together is compared with the behavior of the stack alone in figure below, in which the case effect on the natural frequencies position is evident:




### 7.1.2 Modal Residues Model

The main steps in the construction of the modal residues model are:

- STEP 1. Determination of modal equations of a piezoelectric stack
- STEP 2. Definition of the admittance of the stack in partial fractions form (in frequency domain)
- STEP 3. Introduction of electrical and mechanical losses in the expression of admittance
- STEP 4. Definition of the equivalent electrical circuit parameters

If we see the piezoelectric stack as a two-port system, the electrical port exchanges electrical quantities (voltage and current) while the mechanical port exchanges mechanical quantities (force and velocity). So, for this system we can define four different transfer functions:

- $Z_{VI}$ between current input and voltage output
- $\underline{\underline{Z}}_{\dot{q}F}$ between force input and velocity output

- $\underline{Z}_{VF}$ between force input and voltage output
- $\underline{Z}_{\dot{q}I}$ between current input and velocity output

In this first part of the modal residues model building, we only consider the electrical input and output of the piezoelectric stack, while we suppose that no mechanical load is applied to the stack itself. This means that the extremities of the piezoelectric stack are not subjected to any mechanical force.
Besides, the last two transfer functions will never be considered in our analysis.

$$\begin{bmatrix} V_{pzt} \\ \underline{v}_{pzt} \end{bmatrix} = \begin{bmatrix} Z_{VI} & \cancel{Z_{VF}} \\ \underline{Z}_{\dot{q}I} & \cancel{\underline{Z}_{\dot{q}F}} \end{bmatrix} \begin{bmatrix} I_{pzt} \\ \cancel{\underline{F}_{pzt}} \end{bmatrix} \tag{7.7}$$

In fact, in this first part we are only interested to the electrical impedance, and we build a sort of one-port model (figure 7.3).

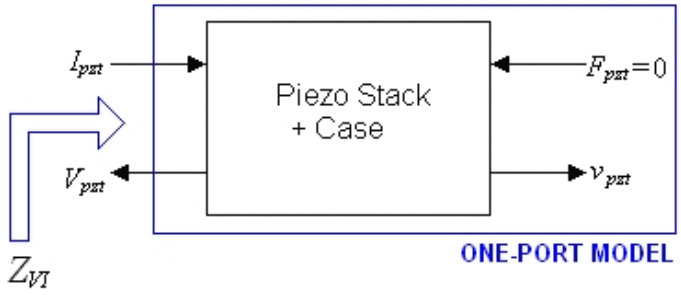



Figure 7.3. One port model of the piezoelectric stack

**Step 1: Determination of modal equations of the piezo stack**

We start from the modal equations 7.5 in which the external force is neglected, and use them to determine an electrical transfer function.
In order to isolate the accelerations vector, the first equation can be divided by $m$:

$$\begin{cases} \ddot{\underline{\zeta}} + \underline{\underline{m}}^{-1}\underline{\underline{k}}\underline{\zeta} - \underline{\underline{m}}^{-1}\underline{\theta}V_{pzt} = 0 \\ CV_{pzt} + \underline{\theta}^T\underline{\zeta} = -Q \end{cases} \tag{7.8}$$

The time derivative of the second equation allows to obtain the current:

$$\frac{d}{dt}(CV_{pzt} + \underline{\theta}^T \underline{\zeta}) = -\frac{d}{dt}Q \rightarrow C\dot{V}_{pzt} + \underline{\theta}^T \dot{\underline{\zeta}} = -I_{pzt} \tag{7.9}$$

At this point, we can convert the system into frequency domain, using Laplace transformation:

$$\begin{cases} s^2\underline{\zeta} + \underline{\underline{m}}^{-1}\underline{\underline{k}}\underline{\zeta} - \underline{\underline{m}}^{-1}\underline{\theta}V_{pzt} = 0 \\ sCV_{pzt} + s\underline{\theta}^T\underline{\zeta} = -I_{pzt} \end{cases} \rightarrow \begin{cases} \left(s^2 + \underline{\underline{\omega_n}}^2\right)\underline{\zeta} - \underline{\underline{m}}^{-1}\underline{\theta}V_{pzt} = 0 \\ s\left(CV_{pzt} + \underline{\theta}^T\underline{\zeta}\right) = -I_{pzt} \end{cases} \tag{7.10}$$

**Step 2: Definition of the admittance of the stack in partial fraction form**

From the first equation, we obtain modal displacements vector $\underline{\zeta}$ and then we can substitute them into the second equation:

$$\begin{cases} \underline{\zeta} = \left(s^2 + \underline{\underline{\omega_n}}^2\right)^{-1} \underline{\underline{m}}^{-1}\underline{\theta}V_{pzt} \\ \left(\underline{\theta}^T \left(s^2 + \underline{\underline{\omega_n}}^2\right)^{-1} \underline{\underline{m}}^{-1}\underline{\theta} + C\right) V_{pzt} = -\frac{I_{pzt}}{s} \end{cases} \tag{7.11}$$

So, the expression $s\left(\underline{\theta}^T \left(s^2 + \underline{\underline{\omega_n}}^2\right)^{-1} \underline{\underline{m}}^{-1}\underline{\theta} + C\right)$ is the admittance of the electric equivalent circuit of the piezoelectric stack, where matrices $\underline{\underline{m}}$ and $\underline{\underline{\omega_n}}$ are modal matrices and so they are diagonal. Considering now each mode as separated from the others, and owing to the matrices properties, we can write the expression of admittance in partial fractions and develop it for the first $n$ modes. The admittance is:

$$Y(s) = \frac{s\theta_1^2}{m_1\left(s^2 + \omega_1^2\right)} + \frac{s\theta_2^2}{m_2\left(s^2 + \omega_2^2\right)} + \ldots + \frac{s\theta_n^2}{m_n\left(s^2 + \omega_n^2\right)} + sC = I_{pzt} \tag{7.12}$$

We can perform a low-frequency analysis ($s \to 0$). The inertial properties of each mechanical parallel branches can be neglected, while elastic contributes are added to the electrical capacitance:

$$C^* = C + \sum_i \frac{\theta_i^2}{m_i \omega_i^2} \tag{7.13}$$

Introducing the residue term $\theta_i^2/m_i = h_i$, the static capacitance (at zero frequency) can be written as:

$$C^* = C + \sum_i \frac{h_i}{\omega_i^2} \tag{7.14}$$

So, the quantity $sC^*$ corresponds to the equivalent low-frequency admittance. Doing then a first-mode analysis, we should detract the first-mode quantity from equivalent low-frequency capacitance:

$$Y(s) = \frac{sh_1}{s^2+\omega_1^2} + s\left[C^* - \frac{h_1}{\omega_1^2}\right]_{s\to 0} = \frac{sh_1}{s^2+\omega_1^2} + s\left[\frac{\left(C^* - \frac{h_1}{\omega_1^2}\right)\left(s^2+\omega_1^2\right)}{s^2+\omega_1^2}\right]_{s\to 0} \tag{7.15}$$

$$Y(s) = \frac{sh_1}{s^2+\omega_1^2} + s\left[C^* - \frac{h_1}{\omega_1^2}\right]_{s\to 0} = \frac{sh_1}{s^2+\omega_1^2} + s\left[\frac{\left(C^* - \frac{h_1}{\omega_1^2}\right)\left(s^2+\omega_1^2\right)}{s^2+\omega_1^2}\right]_{s\to 0} \tag{7.16}$$

Impedance poles are admittance zeros, so the natural frequencies of the impedance are the values that bring the admittance numerator to zero. We keep in consideration the $s$ parameter in the gradient of the curve, so we impose equal to zero just the part in brackets:

$$h_1 + s^2 \left( C^* - \frac{h_1}{\omega_1^2} \right) + \omega_1^2 \left( C^* - \frac{h_1}{\omega_1^2} \right) = 0 \tag{7.17}$$

First zero's pulsation is:

$$s_{z1}^2 = -\frac{h_1 + \omega_1^2 C^* - h_1}{C^* - \frac{h_1}{\omega_1^2}} = -\frac{\omega_1^2}{1 - \frac{h_1}{\omega_1^2 C^*}} \tag{7.18}$$

where $\omega_i$ are pulsations of admittance anti-resonances, which are in correspondence of the impedance resonances.
From experimental tests on a piezoelectric stack or from FEM analysis, we can obtain the value of the first anti-resonance pulsation (which also is the first admittance resonance value) $s_{z1}$. Substituting this value in the last expression, we can determine the first residue $h_1$:

$$h_1 = \omega_1^2 \left( 1 + \frac{\omega_1^2}{s_{z1}^2} \right) C^* \tag{7.19}$$

In this way, we obtain the admittance associated to the first natural frequency:

$$Y_1(s) = \frac{s h_1}{\left( s^2 + \omega_1^2 \right)} \tag{7.20}$$

Keeping then in consideration all the successive modes, we can extend the matter in order to determine all the successive residues. For the first $n$ modes, impedance can be written as:

$$Y(s) = \sum_{i=1}^{n} \frac{s h_i}{s^2 + \omega_i^2} + s \left[ C^* - \sum_{i=1}^{n} \frac{h_i}{\omega_i^2} \right]_{s \to 0} \prod_{j=1}^{i-1} \omega_j^2 \left( \omega_j^2 - s_{zi}^2 \right) \tag{7.21}$$

Residues $h_i$ can be written as:

$$h_i = -\frac{\frac{1}{s_{zi}^2} C^* \prod_{j=1}^{i} \omega_j^2 \left(\omega_j^2 - s_{zi}^2\right) + \sum_{j=1}^{i-1} h_j \prod_{k=2}^{i} \omega_k^2 \left(\omega_k^2 - s_{zi}^2\right)}{\prod_{j=1}^{i-1} \omega_j^2 \left(\omega_j^2 - s_{zi}^2\right)} \tag{7.22}$$

**Step 3: Introduction of mechanical and electrical losses**

In order to obtain a model closer to the reality, it is necessary to introduce a modal damping element into each resonance term of the transfer function. We also keep in consideration a resistance in series with the piezoelectric capacitance, which models the electrical losses due to electrical connections.
Including damping, the resulting admittance is:

$$Y(s) = \sum_{i=1}^{n} \frac{s h_i}{s^2 + 2\zeta\omega_i s + \omega_i^2} + \left( s \left[ C^* - \sum_{i=1}^{n} \frac{h_i}{\omega_i^2} \right]_{s \to 0} \right) \parallel \frac{1}{R_s} \tag{7.23}$$

Damping can be taken into account just near natural frequencies relative to each branch, while at other frequencies it can be neglected.

**Step 4: Definition of the equivalent electrical circuit parameters**

The electric equivalent circuit of the mechanical properties is composed of $n$ parallel resonant branches and the piezoelectric equivalent capacitor $C$. Each resonant branch is constituted by a capacitance and an inductance (in first approximation we neglect losses), so the physical admittance of each branch is:

$$Y_i(s) = \frac{1}{sL + \frac{1}{sC_i}} = \frac{1}{L_i} \frac{s}{\left(s^2 + \frac{1}{L_i C_i}\right)} \tag{7.24}$$

Starting from the admittance obtained from modal analysis 7.21, we should re-write it in the same form of the last expression, in order to do a term-by-term comparison

and determinate the parameters for the electrical equivalent circuit.
Our analysis is limited to the first five most important modes and starting from the admittance related to the first mode 7.20 we can do a term-by-term comparison with the parametric admittance of the first parallel branch, that is equal to:

$$Y_1(s) = \frac{1}{L_1} \frac{s}{\left(s^2 + \frac{1}{L_1 C_1}\right)} \tag{7.25}$$

We obtain the following relations, which can be used in order to define lumped parameters values for the electrical equivalent circuit:

$$\begin{array}{ll} h_1 = \frac{1}{L_1} & L_1 = \frac{1}{h_1} \\ & \rightarrow \\ \omega_1^2 = \frac{1}{L_1 C_1} & C_1 = \frac{1}{L1\omega_1^2} \end{array} \tag{7.26}$$

All the relations before used to evaluate the parameters relative to the first branch can be extended to each successive branch of the electrical equivalent circuit:

$$\begin{array}{ll} h_i = \frac{1}{L_i} & L_i = \frac{1}{h_i} \\ & \rightarrow \\ \omega_i^2 = \frac{1}{L_i C_i} & C_i = \frac{1}{Li\omega_i^2} \end{array} \tag{7.27}$$

Using residues value, we can characterize the inductive and the capacitive elements of the electrical equivalent implementation, while resistance value is determined by comparison with experimental data obtained on piezoelectric stack sample.

## 7.2 FEM model validation

### 7.2.1 Experimental Measurement of the Piezoelectric Stack Impedance

Measures are obtained using a HP4192A LF Impedance Analyzer 5Hz-13MHz where the system is mechanically unconstrained and voltage driven.

Three measures were performed:

- Resistive behavior measure in the range $0.1KHz \div 9KHz$ (figure 7.4)
- Capacitive behavior measure in the range $0.1KHz \div 9KHz$ (figure 7.5)
- Impedance measure in the range $0.1KHz \div 100KHz$ (figure 7.6)

The measured impedance is then used to validate the FEM model and to identify damping of residues model.

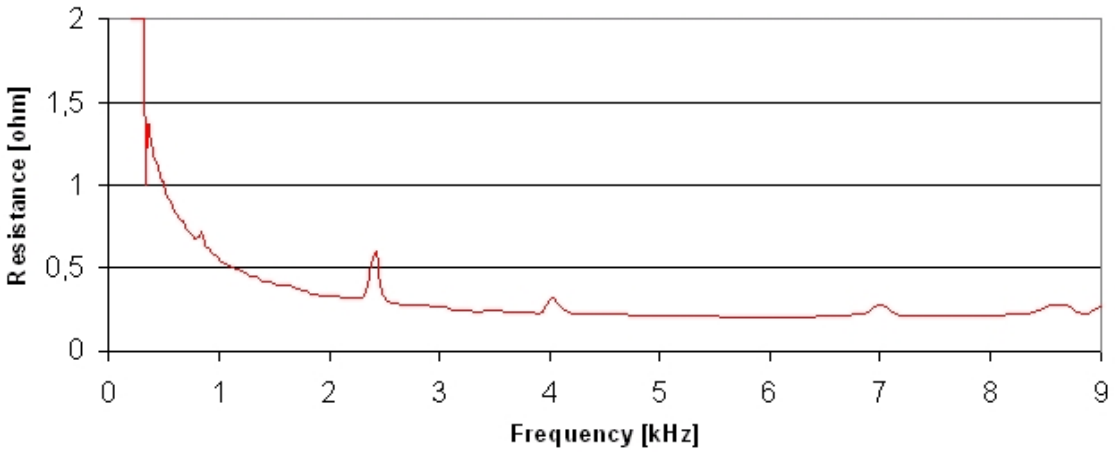



Figure 7.4. Resistive behavior measure of a piezoelectric stack

A graphical comparison between the results of the FEM model and the experimental measurements is shown in figure7.7.
It can be noticed a correct fit of the first and second natural frequencies, while the third natural frequency has not been identified. This is probably due to a resonance effect of the structure that has not been modeled.
For the same reason also the other natural frequencies are shifted with respect to those identified experimentally.

### 7.2.2 Electrical Equivalent Circuit

The admittance can be represented as the following electrical equivalent circuit where we have included parallel branches that correspond to the first 5 modes of the stack (electrical equivalent of mechanical modes)(figure 7.8).

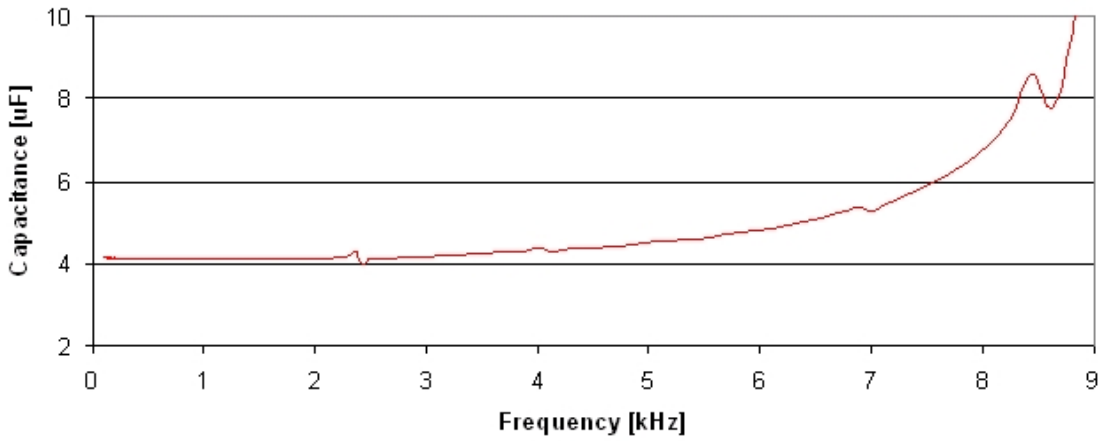



Figure 7.5. Capacitive behavior measure of a piezoelectric stack

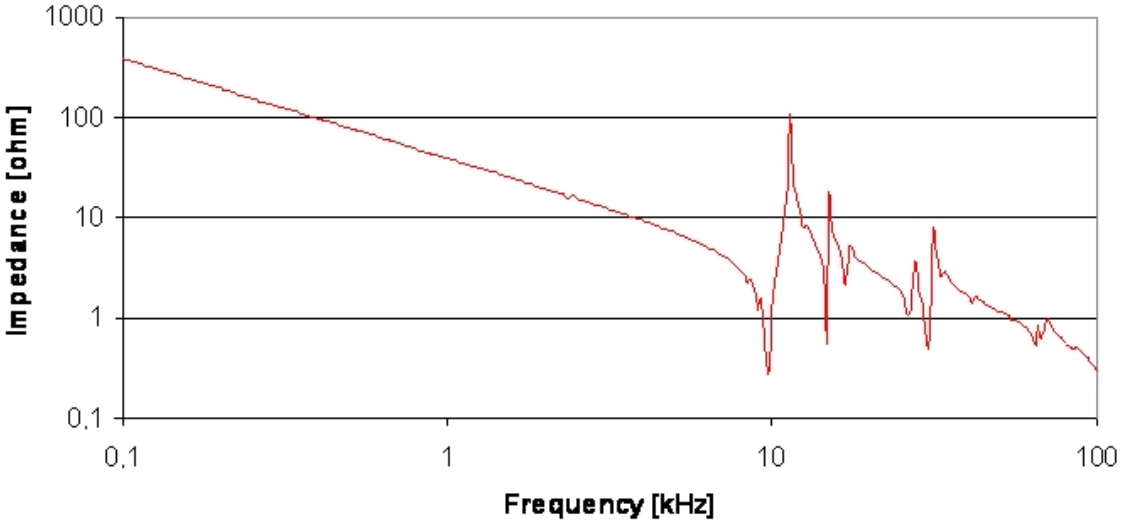



Figure 7.6. Impedance measure of a piezoelectric stack

Starting from this circuit, the computed impedance has been compared to the experimental impedance. In the figure 7.9 it can be seen this comparison.

The electrical admittance is a good approximation of the electrical behavior of the piezoelectric stack , so it will be used as a power driver load in order to design it.

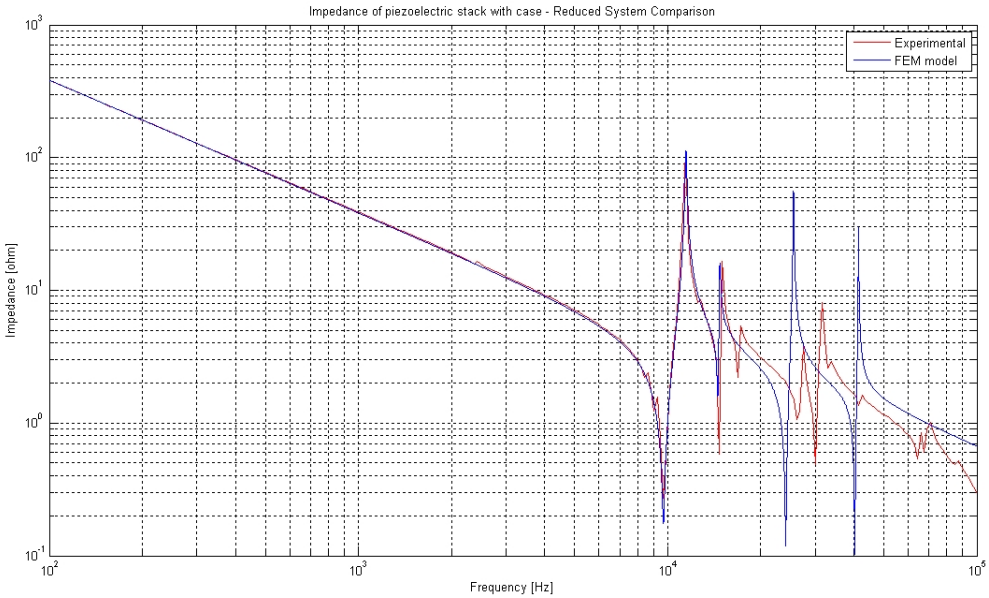



Figure 7.7. Impedance measure vs FEM model

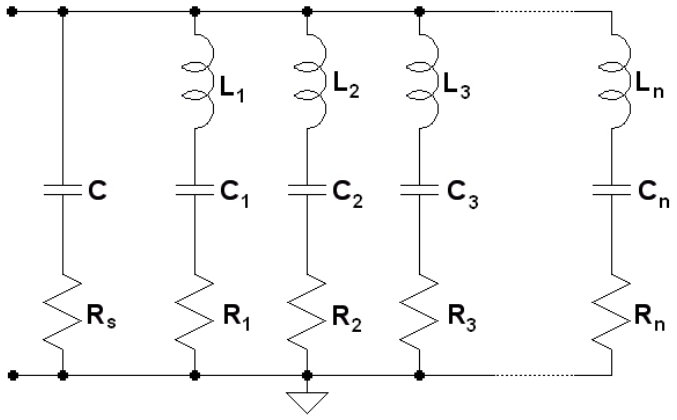



Figure 7.8. Electrical equivalent circuit of the piezoelectric stack

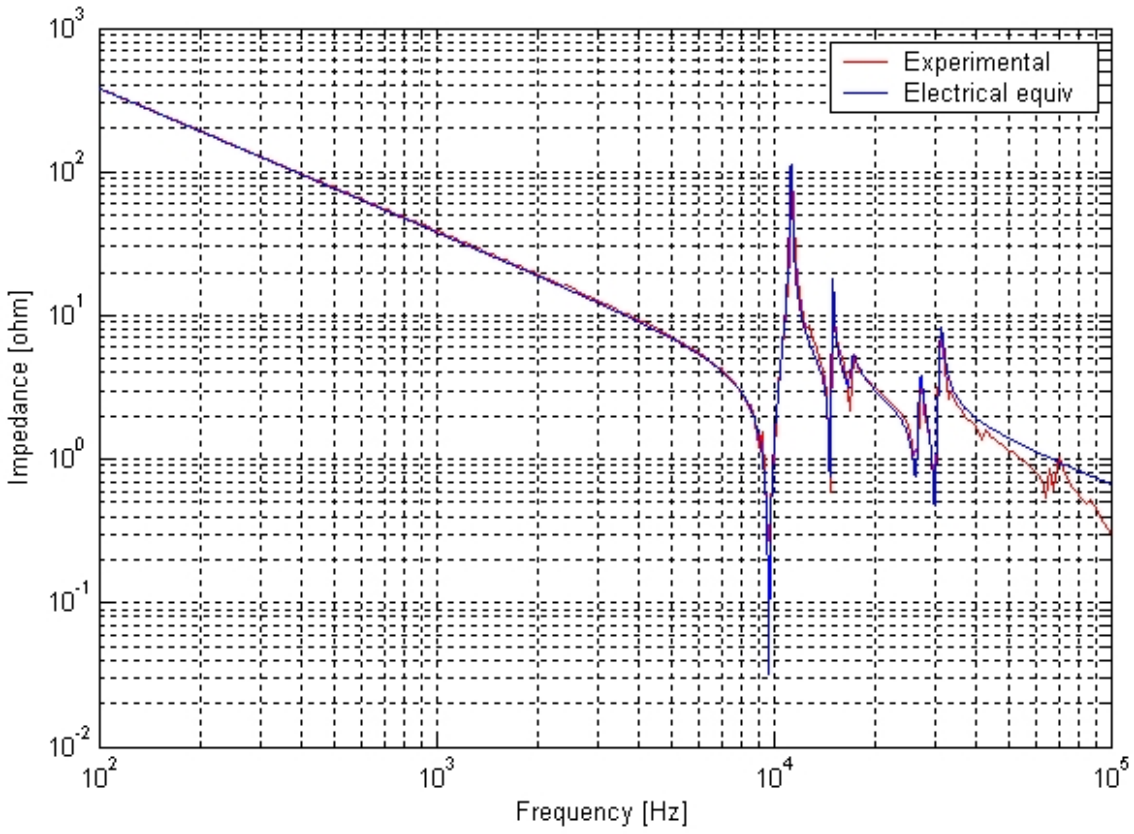



Figure 7.9. Experimental impedance vs electrical equivalent circuit model

# Chapter 8

# Design

Piezoelectric actuators controller is aimed at the control of the `tip` displacement, where the `tip` is the free end of the piezoelectric stack.
Unfortunately, no velocity or position or force measures can be performed neither inside the actuator nor on the piezoelectric stack in a real application.
So, in order to estimate the actuator tip position we need to measure a related quantity. From the constitutive equations of a piezoelectric stack, we can see that the tip position is strictly related to the electrical charge loaded in the equivalent capacitance of the actuator itself.
A direct charge measurement is also possible but it is an intrusive technique. However, loaded charge is proportional to the current transferred to the piezoelectric stack by the Power Driver.
The control is performed with two feedbacks:

1. An inner loop which controls the load current

2. An outer loop which controls the charge stored in the actuator

Hence from the charge accumulated in the piezoelectric equivalent capacitance it is possible to estimate the needle position.

## 8.1 Charge Estimation

Charge can be estimated by:

- Analog integration of the measured current

- Digital integration of the measured current

### 8.1.1 Analog integration of measured current

We can use an analog integrator and integrate the current given to the load, and thus obtaining the charge stored in it (see figure 8.1). The charge value acquired in real time is used in the charge outer loop while the value sampled at the end of the charging phase allows to calculate the energy given to the load (and eventually the capacity). The analog integrator must be reset by the digital system before each charging phase to avoid the continuous integration of eventual current offset. Energy evaluation at the end of each cycle lets the controller device have parameters correction before successive charging phase, in case of capacity variation due to temperature.

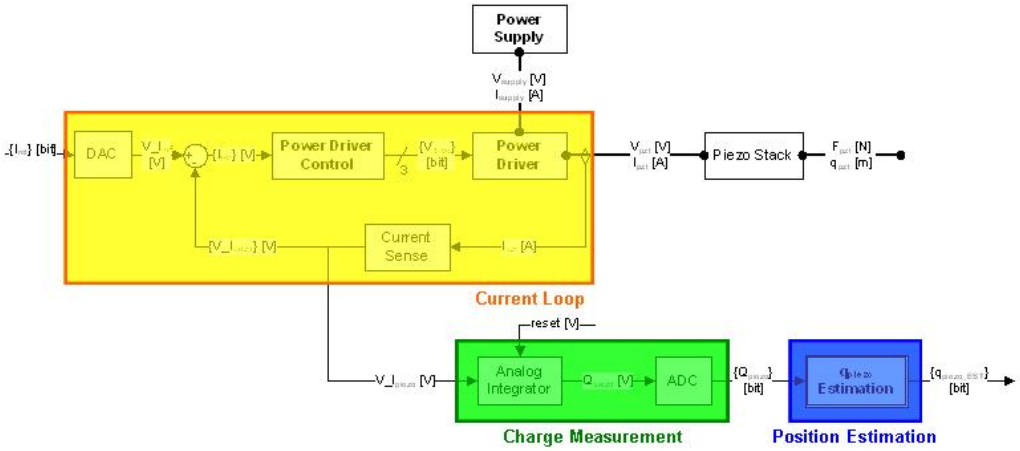



Figure 8.1. High level block scheme with analog current integration

### 8.1.2 Digital integration of measured current

Current flowing through piezoelectric can be sampled and converted at high frequency; hence we can numerically evaluate the numeric integral of the current (see figure 8.2). To perform a valid integration the sampling period must be constant; in this way the integral can be implemented with the discrete Tustin method.
In order to reduce the switching noise, the current sampling needs to be performed synchronously with the MOS switching but shifted in time with respect to the closing/opening instants of the power switches.
If the switching frequency is not constant, for example implementing a hysteretic current control, the digital integration is very difficult to be performed.

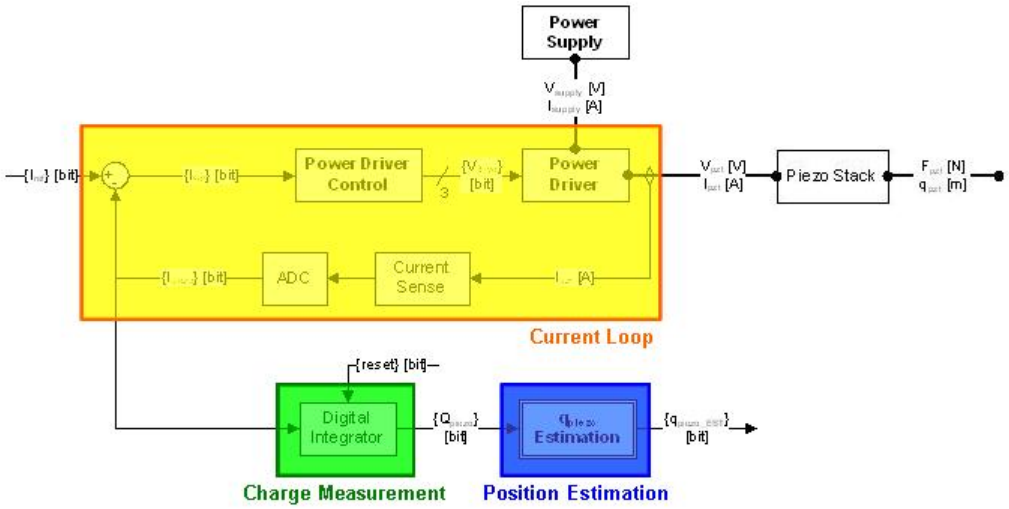



Figure 8.2. High level block scheme with acquisition and digital current integration

## 8.2 Power Driver

Considering the piezoelectric stack model previously obtained, for power driver design we consider just the electrical behavior:

- we are not interested in the mechanical effects, so we do not consider the parallel branches of the model (which represent the mechanical vibrating modes)
- Mechanical effects will be introduced only in a second time to validate the power driver behavior

In addition to the piezoelectric equivalent circuit we put also a parallel resistance which is effectively present in the real system because of safety reasons (figure 8.3).

### 8.2.1 Concept

Analyzing the constitutive equations of a piezoelectric stack actuator, we can see that the electric equivalent circuit is basically made of a capacitance.
A good method to drive a capacitive load is to use a current generator driven by an external signal: the simplest way to realize this type of current generator is to use a trans-conductance amplifier (that accepts a voltage input and gives a current output) and driving it using a PWM signal (figure 8.4).

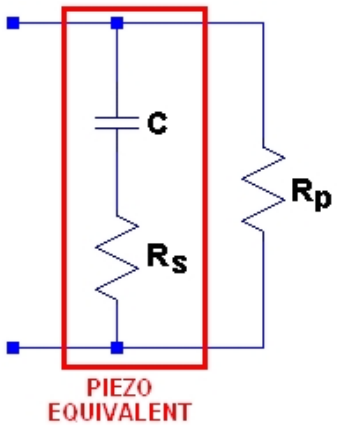



Figure 8.3. Parallel resistance present on the piezoelectric stack under test

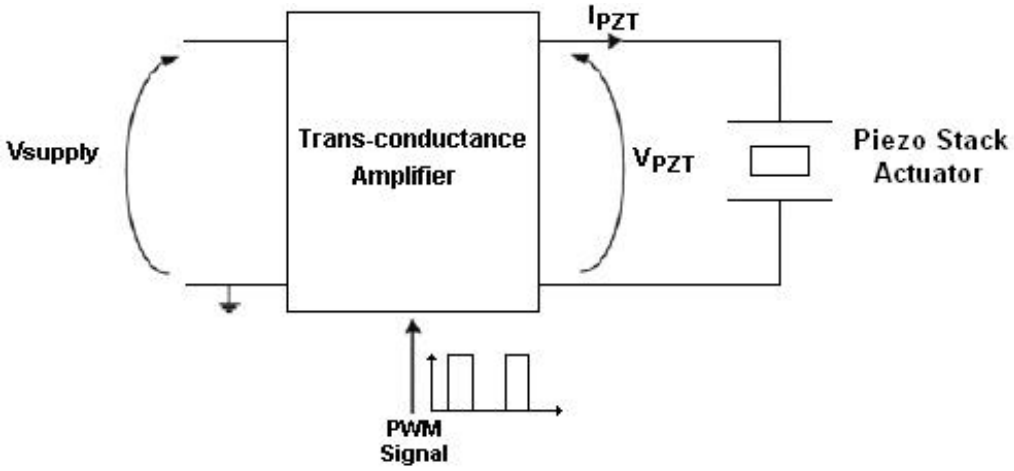



Figure 8.4. Idea of the trans-conductance amplifier driven by a PWM signal

One solution for trans-conductance amplifier is to use a switching converter.
The power driver shall be able to charge and discharge the piezoelectric element, so the switching converter must be bidirectional in current.
To realize a good current generator and to evaluate energy, the current mode control is preferable: using this control strategy, the output current follows a given current profile.

### 8.2.2 Electrical Specifications

From technical specification about the piezoelectric stack actuators, the power driver module shall be able to drive a load with the characteristics and operative conditions below:

- Typical capacitive load of $3\mu F \div 10\mu F$ (depending on temperature)
- Absolute maximum output voltage is $160V$
- Maximum peak current is limited at $15A$
- Maximum current slope is $2A/\mu s$

### 8.2.3 Topology

The driver topology chosen is Bidirectional Buck converter, operating in continuous conduction mode (CCM) in order to reduce current stresses on electronic components.
In addition to the basic structure, we put a dissipative section (a sort of braking transistor) to obtain the complete discharge of the piezoelectric capacitance when needed.
The most important variable for the piezoelectric actuators is the charge accumulated in the equivalent capacitance, which depends on temperature. So we need to measure and control the current absorbed and supplied by load, and also the related voltage on it.
Using a current mode control all system poles become real and one of them goes to high frequency avoiding resonances given by load inductance and capacitance. Moreover, this control strategy gives the advance of switch protection in case of output short circuit.
The basic converter scheme is shown in the figure 8.5.

The value of inductance depends on both maximum current slew-rate of the piezoelectric actuator and DC bus voltage. Input capacitance is calculated considering the effective energy that must be supplied in a single charge/discharge phase. Other components are chosen considering the absolute maximum electrical stress.

### 8.2.4 Switches Driving Strategy

Bidirectional buck is driven acting, correctly, on the opening-closing phases of switches M1 (high side), M2 (low side) and M3 (brake MOS) (figure 8.6).
The strategy is to drive both the transistors in a complementary way, with a short

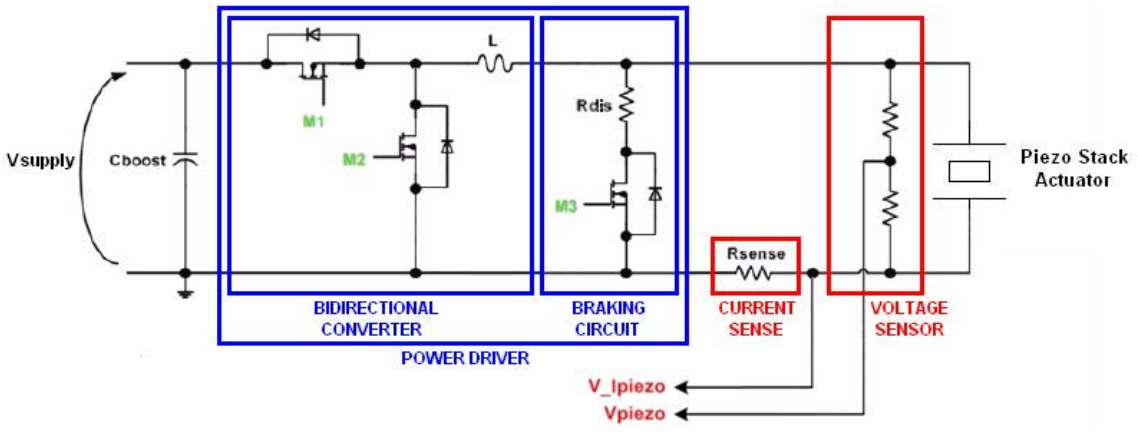



Figure 8.5. Bidirectional buck converter used in the driving module topology

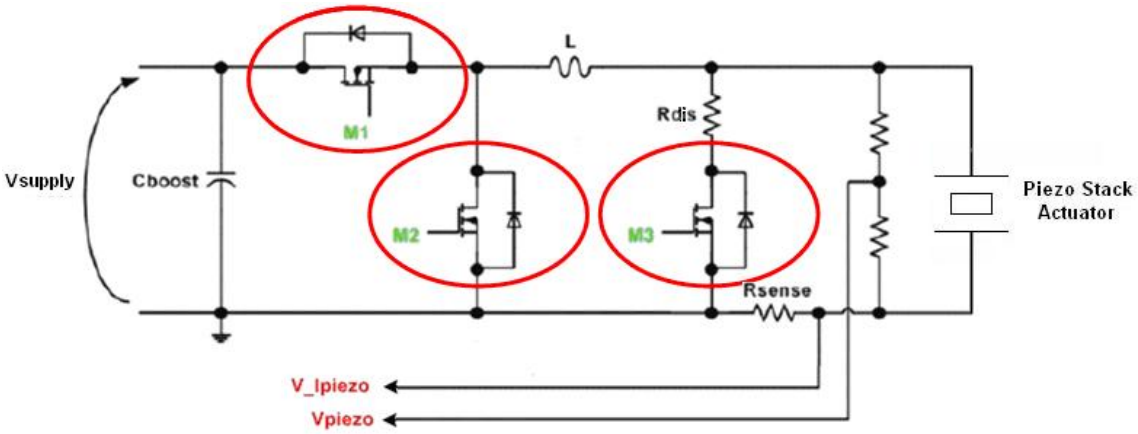



Figure 8.6. High side MOS (M1), low side MOS (M2) and brake MOS (M3) of the power circuit

dead band time in order to avoid cross conduction. Using this strategy it is possible to have current flow inversion through the inductor. This method is called synchronous rectification.
During the discharge phase, when the piezoelectric voltage reaches an established value the control turns on the brake transistor for the complete discharge of the load.
In the figure 8.7 are shown the voltage drop on piezoelectric stack, current flow on the load with respect to the power switches phases.

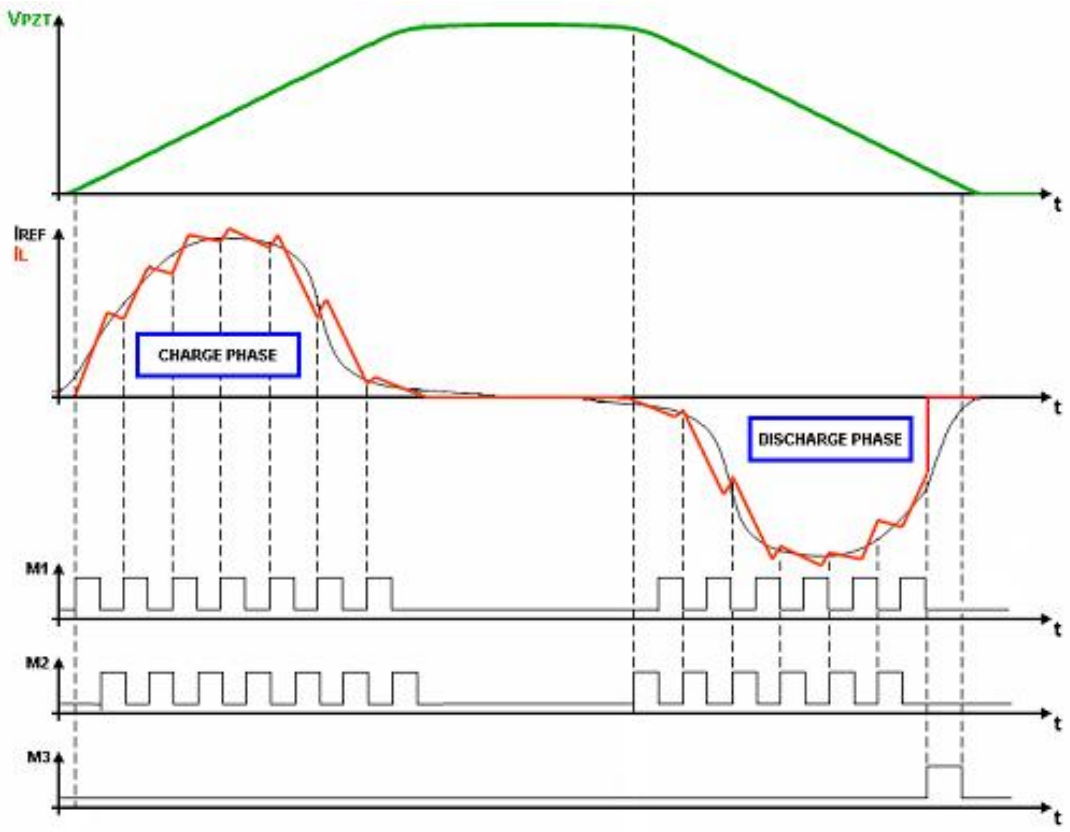



Figure 8.7. Voltage drop and current flow on piezoelectric load in synchronous rectification method

## 8.3 Current Control

The first step is to control the Bidirectional Converter using a peak current mode control: the peak method of inductor current control works by comparing the upslope of inductor current with a current peak reference set by the digital platform. The comparator turns the power switch off when the instantaneous current reaches the desired level (see figure 8.8).

The current ramp is usually quite small compared to the reference level, especially when $V_{IN}$ is low. As a result, this method is extremely sensitive to noise. A peak of noise is generated each time the switch turns on. A fraction of a volt coupled into the control circuit can cause it to turn off immediately, resulting in a sub-harmonic operating mode with much greater ripple. The peak current mode control method is

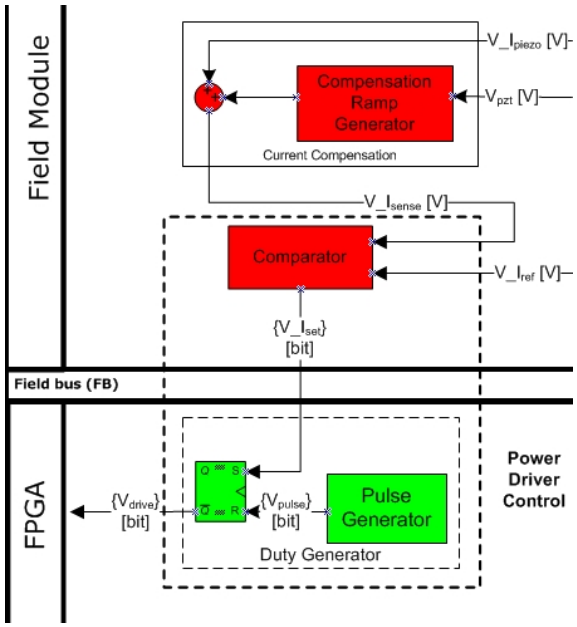



Figure 8.8. Current control simplified scheme

inherently unstable at duty ratios exceeding 0.5, resulting in sub-harmonic oscillation. This happens when the output voltage becomes greater than half of the supply voltage. Sub-harmonic instability occurs only in continuous conduction mode. Standards methods to avoid this phenomenon include a compensation ramp, to be added to the requested current level. Other solutions are possible. For our circuit the following solutions can be considered:

- Peak current with analog compensation ramp
- Solution omitted (patent in progress)
- Peak-Valley current mode
- One Cycle Control (OCC)
- Average current mode

### 8.3.1 Peak current with analog compensation ramp

Consists in a time continuous compensation ramp added to the measured current signal. Ramp is generated by a passive integration of the output voltage, and added via a passive (resistive) network.

### 8.3.2 Section Omitted

For confidentiality reasons, due to patents in progress on the part of the private company that collaborates on this project, this section has been omitted.

### 8.3.3 Peak-Valley current mode

Sub-harmonic instability occurs at duty cycle $> 50\%$ for peak current mode control. If valley current mode control is used, sub-harmonic instability occurs for duty cycle $< 50\%$. It is possible to change the control strategy when the duty cycle crosses the 50% boundary.
However at the end of the charge phase, converter runs in discontinuous conduction mode (DCM), and valley current control cannot be used. So the controller should recognize when the duty cycle becomes greater than 50% in order to revert to peak current mode control.

### 8.3.4 One Cycle Control

The `One Cycle Control` technique was developed as a general pulse width modulator control method. It is also known as the integration-reset technique wherein the key element is the resettable integrator. The OCC uses the pulsed and non-linear nature of switching converters to achieve instantaneous control of the average value of the switching voltage or current. This technique is designed to control the duty cycle in real time such that in each cycle the average of the chopped waveform is exactly equal to the control reference

### 8.3.5 Average current mode

This method uses a standard PWM modulator, and has no sub-harmonic instability; however it introduces some delays (narrow bandwidth system).

# Chapter 9

# Implementation

Based on considerations explained in chapter 8, it is necessary to have a power driver capable of limiting the absolute maximum current value because there is a fixed operating current range as seen in 8.2.2.
From constitutive equations of piezoelectric material we can see that displacement is strictly related to the charge stored on piezoelectric actuator, so the power driver must control the amount of charge supplied.
Hence, power driver module is made up of a bidirectional converter and two feedback loops that allow controlling both the charge stored and the current that flows in the piezoelectric actuator.

## 9.1 Inner loop

As mentioned in 8 the control architecture is composed of an inner loop which controls and limits the maximum peak current starting from a current reference generated by an outer loop that guarantees the correct amount of charge stored into the piezoelectric load.
The system architecture including the control loops is illustrated in the following technological scheme (or layout) (figure 9.1).

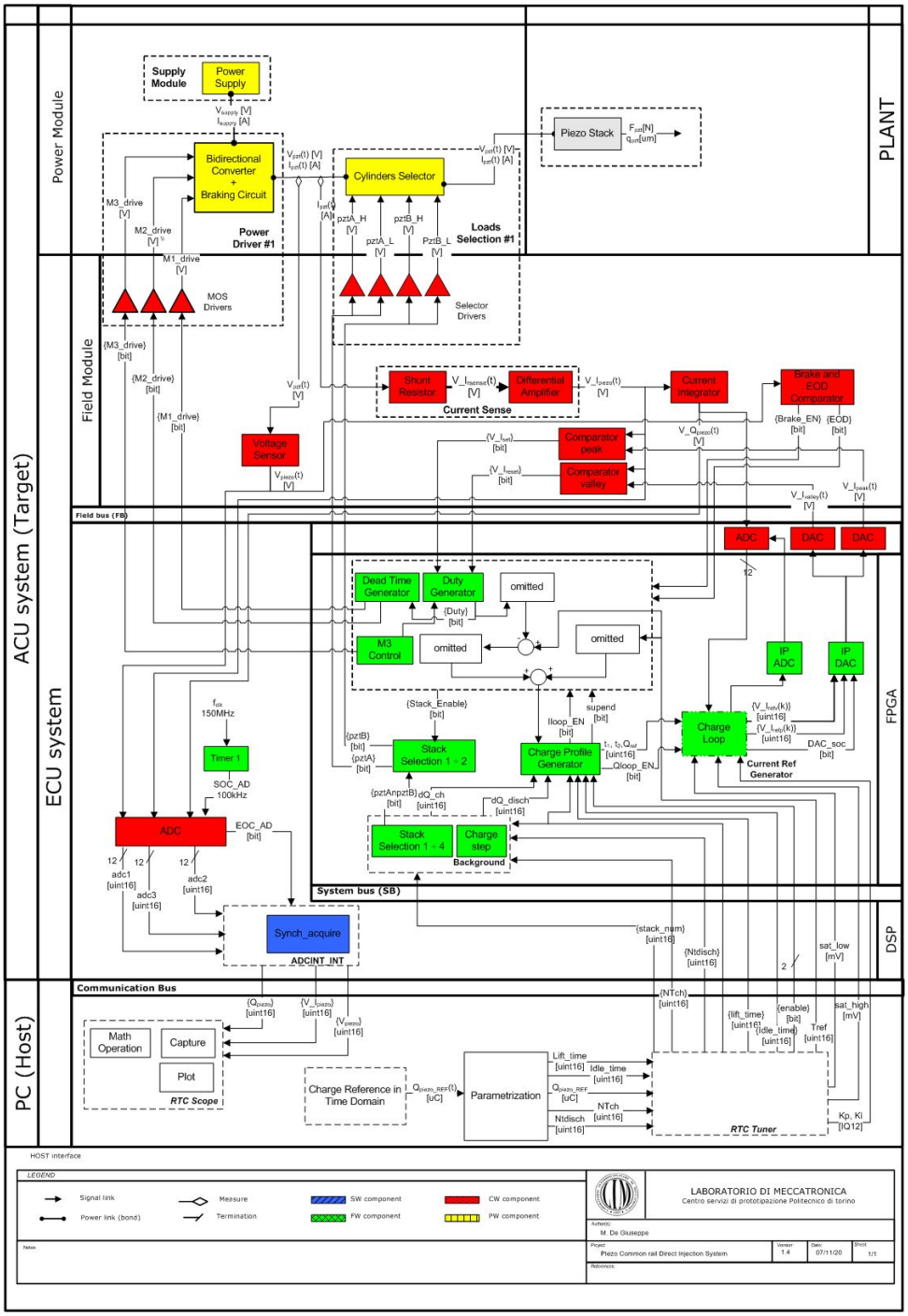



Figure 9.1. Technological scheme of the system

The Actuation Control Unit (ACU) can be implemented with a prototyping stack consisting on two boards interconnected by means of the Field Bus (including both analog and digital signals)(figure 9.2):

- ECU, which is the Electronic Control Unit based on a DSP and a FPGA: the core logic on which runs Firmware and Software
- Field Module, containing Conditioning and Power circuits

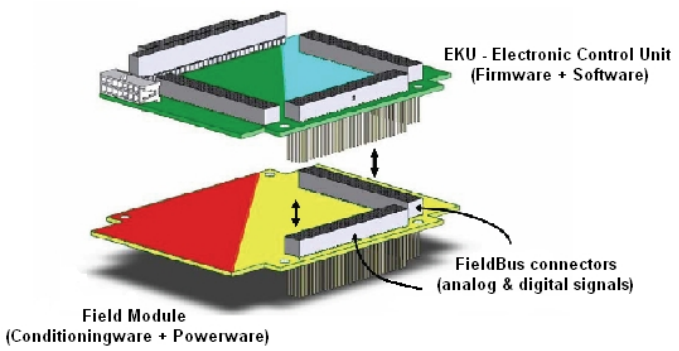



Figure 9.2. Basic prototyping stack

### 9.1.1 Current Loop

This feedback loop represents the low level control of the bidirectional converter and the implementation is distributed between firmware and hardware.
It is based on peak current mode control of a switching regulator, where the controller commands directly the peak current in the inductor of the power converter. Hence the inductor current follows instantaneously the control input.
Current is sensed and conditioned using a sense resistor and a differential amplifier with an input bridge in order to reject the common mode and filter the high frequency noise in differential mode.
An analog comparator is used in order to have a continuous comparison between reference current and inductor one.
Digital current control is based on three main blocks that are implemented as IP-core on FPGA:

- Duty cycle generator
- Dead band time generator

- M3 (brake switch) control

This solution is based on a fixed frequency control where the switching frequency is a design parameter.
Variable frequency is undesirable since it leads to an over dimensioning of the output filter, increases the switching losses or the ripple current in the inductor.
Reference current is generated digitally by the outer loop and converted using a DAC converter managed by FPGA.
Current loop is highlighted in figure 9.3.

A simplified block diagram representation of current loop is shown in figure 9.4.

The input to the current loop is the set-point current $V\_I_{ref}$ (express in voltage) which is compared to the sensed inductor current $V\_I_{piezo}$ and sets the M1 duty cycle M1_drive. The duty cycle is passed to the power driver stage (switching elements, inductor, and output capacitance), which produces a corresponding inductor current. The inductor current is fed back through a sensing gain (current sense) and returns to be compared with $V\_I_{ref}$.
Second step is to substitute the peak current mode, which is inherently unstable (sub-harmonic instability) with the hysteretic current mode. This control mode is stable but the switching frequency is variable depending on the voltage drop on the load. In order to limit this phenomenon, different methods can be implemented.

### Current control

The peak current control has an intrinsically non linear control law because the duty cycle is generated starting from a comparator and a digital device (FF-SR theoretically) that are two non linear blocks.
The power switch M1 (high side) turns on when the clock pulse sets the ideal FF-SR. Then the inductor current and the switch current begin to increase. The FF-SR is reset, and the power switch is turned off, when the comparator detects that the sum of the inductor current has increased to the value commanded by the control signal. After a dead time the low side MOS M2 is turned on; it is turned off when clock pulse sets newly the ideal FF-SR.
When the system is in charge phase the controller works in this way, while the controller works in a complementary mode during the discharge phase: the output of comparator is normally high and the power switch M2 is turned on. The FF-SR is reset and M2 is turned off when current measured is equal to the set-point value current. After a dead time the high side MOS M1 is turned on and turned off at new switching cycle.

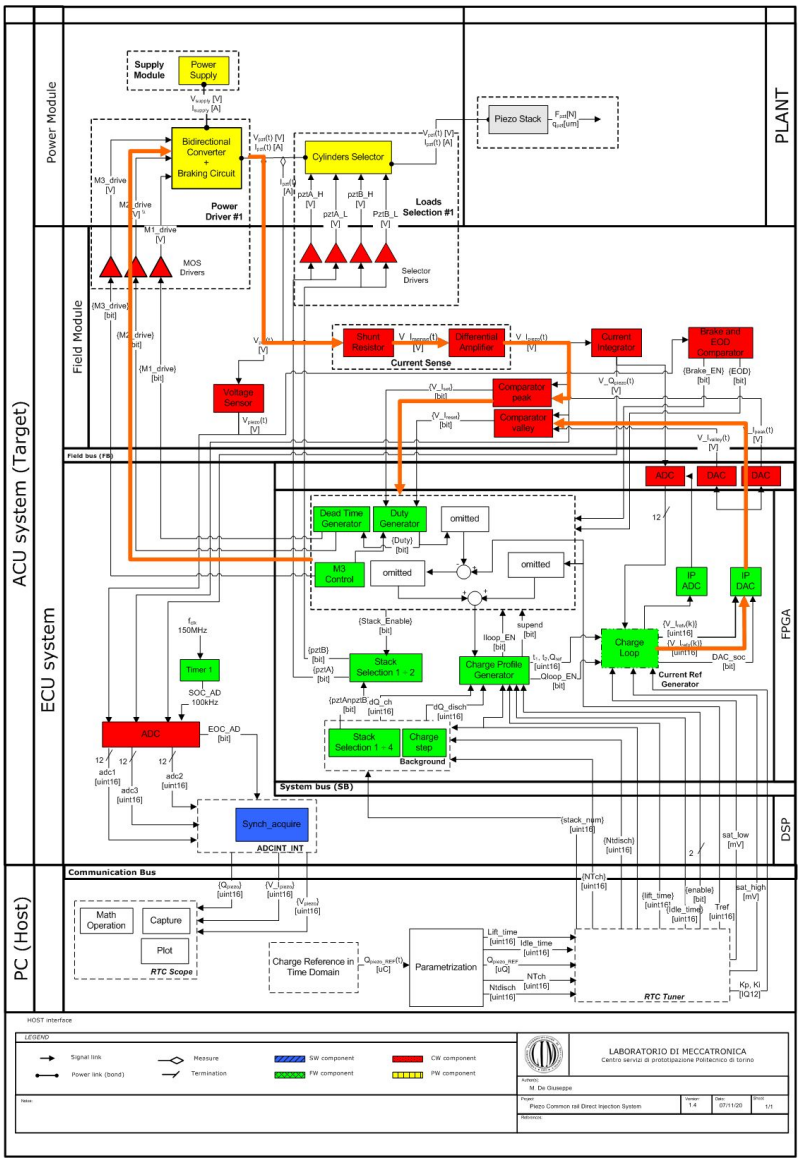



Figure 9.3. Highlighted current control loop on technological scheme

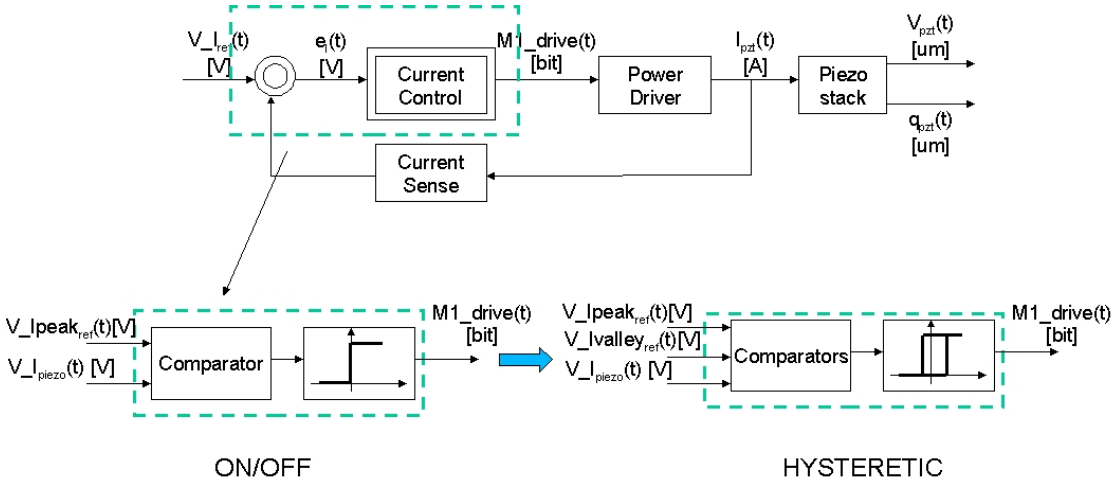



Figure 9.4. Block diagram of the current control loop

In the hysteretic current control the clock pulse setting the FF-SR is substituted by another comparator which inputs are the valley current reference and the measured current. In this way the current is held between two reference values set by the outer loop.
It is possible to implement three different types of MOS driving strategy in order to improve the efficiency of power driver:

- standard method: drives M1 during charge phase and both M2 and M3 during discharge phase (M3 for complete discharge)
- synchronous rectifier: drives M1 and M2 in both charge and discharge phase and M3 for complete discharge
- mixed solution: starts with synchronous rectifier and when current get to be negative it passes in standard method

The Switching driving method currently implemented is a synchronous rectification where M1 and M2 are driven both during charge and discharge phase and M3 for complete discharge of piezoelectric element.

Key advantages with current mode control are:

- excellent line regulation
- simple compensation design
- robustness to large load variations

- inherent cycle-by-cycle current limiting

#### Current sense

It uses the low side principle, in which the sense resistor is connected in series with the ground path. To reduce the undesirable resistance in the ground path we have chosen a small value resistor.
To condition the current monitoring we have used a differential amplifier with a fixed voltage gain: this block reduces the common mode and limits the band of current sensor.

#### Power driver

As previously mentioned the power driver is a Buck bidirectional converter that allows to charge and discharge completely the piezoelectric actuator.
If we control the power driver using a current mode this stage becomes a voltage controlled ideal current source.

#### Piezoelectric stack

Piezo stack block represents the constitutive equations obtained using FEM model, hence:

- Ordinal Differential Equations
- External force is neglected
- Small signal model
- Obtained at ambient temperature

Input of piezoelectric stack is the current supplied by power driver while the outputs are the tip displacement and voltage drop on it.

### 9.1.2 Section Omitted

For confidentiality reasons, due to patents in progress on the part of the private company that collaborates on this project, this section has been omitted.

### 9.1.3 Current Loop adaptation on $V_{pzt}$

Our goal in this project was to design a power module able to discharge completely the residual piezoelectric element charge and to recover as much energy as possible. Brake transistor is used to completely discharge the piezoelectric actuator.
When the output voltage is below a fixed threshold, current control turns on the brake transistor until the voltage drop on piezoelectric stack is approximately zero (there is a second threshold that turns off the brake transistor).
While the brake transistor is turned on, the two switches of the half bridge are turned off.
The block diagram of this adaptation of current control is shown in figure 9.5 while the current loop adaptation on voltage drop on piezoelectric actuator is highlighted in figure 9.6.

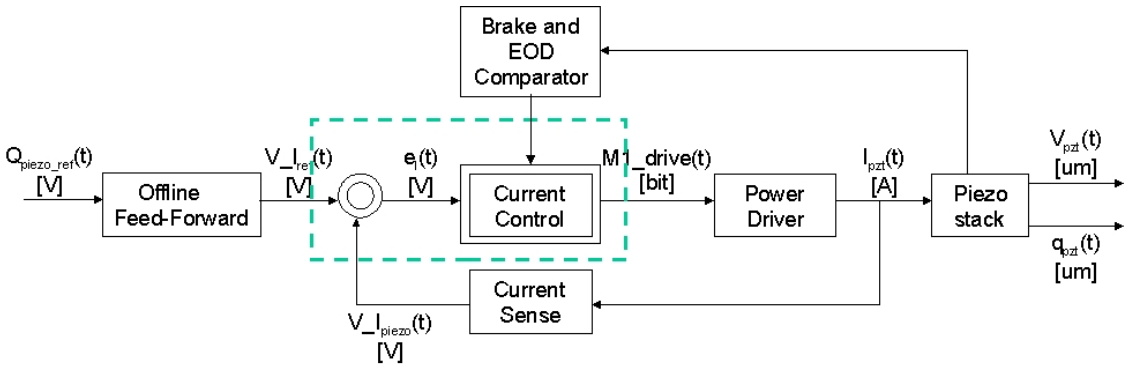



Figure 9.5. Block diagram of the current control loop adaptation on voltage drop

Brake and EOD (End Of Discharge) comparator block acquires the voltage drop on piezoelectric stack and return a digital command that disables the switching conversion and turns on and off the brake comparator (M3).

## 9.2 Outer loop

### 9.2.1 Charge Loop

As mentioned before, the system includes an outer loop that allows controlling the amount of charge stored on piezoelectric actuator.
Since the bidirectional converter is a step-down during charge phase and a step-up in discharge phase, it has a different behavior depending on a specific operative phase. Therefore the equivalent transfer function of the system with a current feedback

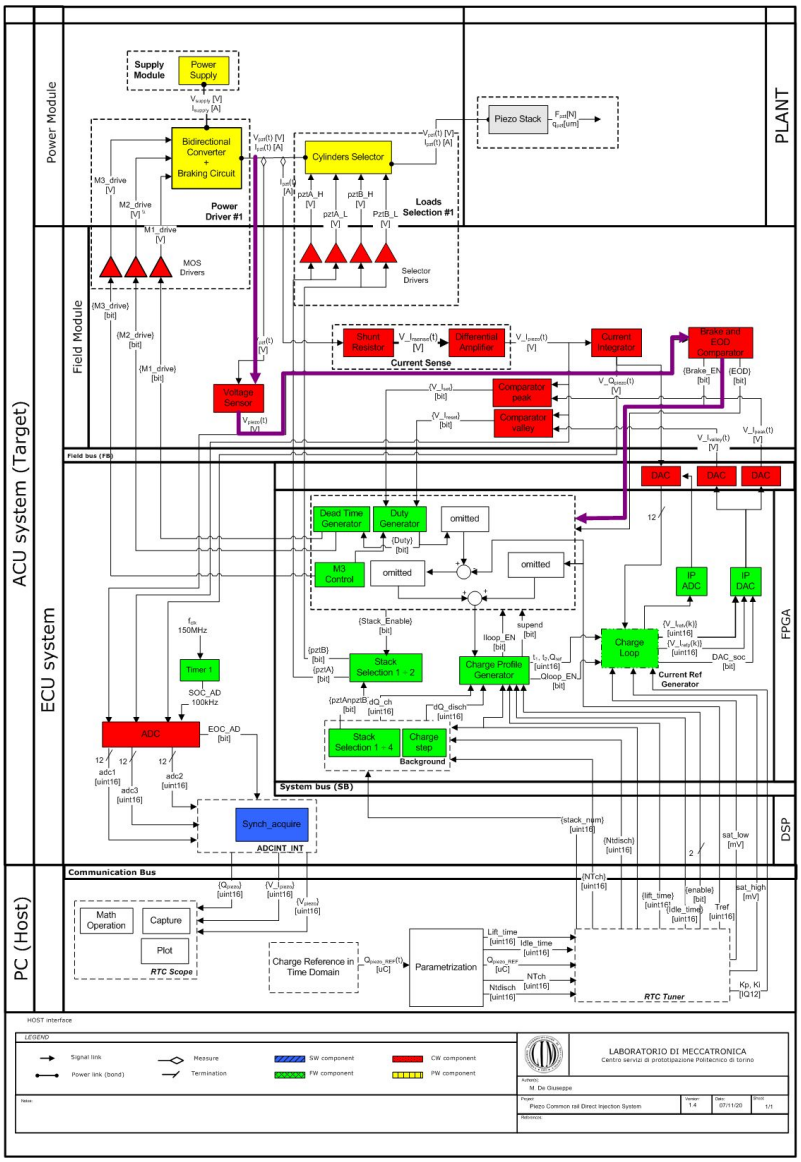



Figure 9.6. Highlighted current loop adaptation on voltage drop

depends on operative condition.
So, we have implemented an outer loop with the same control law for both phases, but with different control parameters.
It is a generic PI control loop feedback that attempts to correct the error between a measured charge and a desired set point by calculating a corrective action that can adjust the process accordingly.
This PI controller algorithm involves two different parameters:

- Proportional: determines the reaction to the current error
- Integral: determines the reaction based on the sum of previous errors

This control technique consists of an ADC converter, a processing unit that implements a control law (DSP or FPGA) and a sensor that measures the variable that must be controlled.
We have implemented a discrete PI controller that gives the output at a fixed sampling frequency.
The PI controller is shown below, where $Tp$ and $Ti$ denotes the time constant of the proportional and integral terms.
The transfer function of the system is:

$$\frac{u(s)}{e(s)} = K_p \left(1 + \frac{1}{T_i s}\right) \tag{9.1}$$

This gives the command $u$ with respect to the error $e$ in the time domain:

$$u(t) = K_p \left(e(t) + \frac{1}{t_i} \int_0^t e(\tau) d\tau\right) \tag{9.2}$$

We can now approximate the integral term to get the discrete form:

$$\int_0^t e(\tau) d\tau \approx T \sum_{k=0}^{n} e(k) \tag{9.3}$$

Where $n$ is the discrete step at time $t$ and $T$ is the sampling period.
In this way the controller become:

$$u(n) = K_p e(n) + K_i \sum_{k=0}^{n} e(k) \tag{9.4}$$

Where:

$$K_i = \frac{K_p T}{T_i} \tag{9.5}$$

Directly measure charge storage is possible but only in intrusive form, so it is necessary to obtain charge value using an indirect measurement starting from the inductor current.
Therefore to measure the charge stored on piezoelectric actuator we have used an analog integrator with reset at the end of each charge phase.
A complete charge loop with inner loop and current control adaptation is shown in figure 9.7 where the direct charge measure link has been highlighted as well.

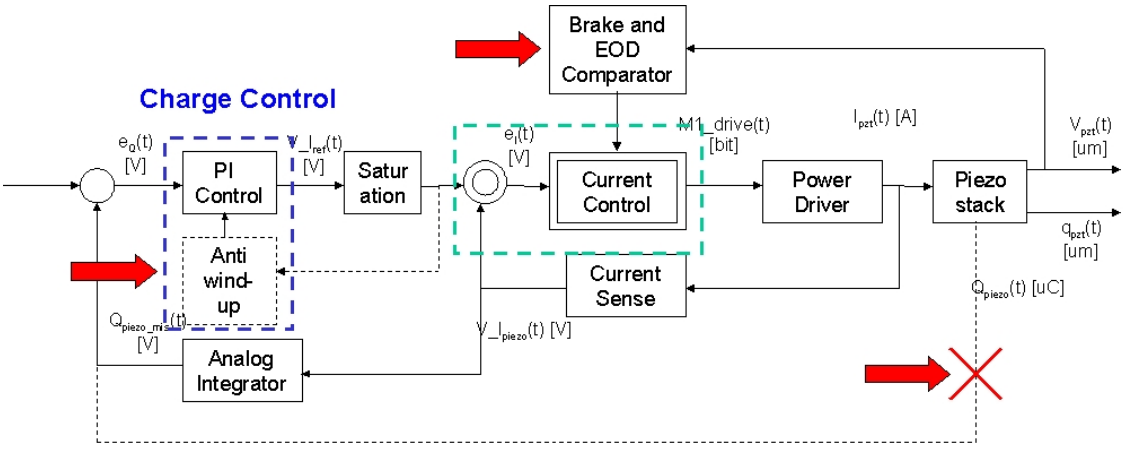



Figure 9.7. Block diagram of the charge control loop

### Charge Control

This block represents the PI algorithm implemented as IP-core on FPGA. It is computed at a fixed frequency ($100KHz \div 166KHz$)and returns the set point current

value to apply at current loop, starting from the error between measurement and reference.
The anti-windup implementation is very simple but efficient: if the command at the previous sampling step is saturated, the integral branch of the compensator is not executed; in this way the pre-saturation command does not rise too much over the saturation limit and the command is fast to recover when the proportional branch becomes negative.
This anti-windup control is clearly not linear; the implementation on the DSP can be onerous due to the `IF` instruction; on the FPGA instead, the `IF` instruction does not take more time because is executed in parallel with the rest of the code.

#### Analog integrator

It is the analytical model of analog integrator utilized to obtain the charge stored starting from the current supplied by power driver.
Charge is obtained using this expression:

$$V\_Q_{piezo} = \frac{A_V R_{sense}}{RC} K_{ADC} \int_0^t i_L d(t) = \frac{A_V R_{sense}}{RC} K_{ADC} Q_{piezo} \tag{9.6}$$

Where:

- $V\_Q_{piezo}$ is the output of ADC converter
- $A_V$ is voltage gain of current sense differential amplifier
- $R_{sense}$ is the value of sense resistance
- $K_{ADC}$ is the ADC conversion ratio
- $R$ and $C$ are components of analog integrator in order to obtain a desired time constant

### 9.2.2 Section Omitted

For confidentiality reasons, due to patents in progress on the part of the private company that collaborates on this project, this section has been omitted.

### 9.2.3 Tuning of the compensator parameters

The mathematical model of the system is used to find the PI parameters. They are computed in order to obtain the desired response. The accurate mathematical description of the system is rarely available, thus experimental tuning of the PI parameters has to be performed.
Several methods for tuning the PI loop exist. The choice of method will depend largely on whether the tuning process can be taken off-line or not. Ziegler-Nichols method is a well-known on-line tuning strategy.
The first step consists in setting the integral gains to zero, increasing the proportional gain $K_p$ until a sustained and stable oscillation is obtained on the output. Then the critical gain $K_c$ and the oscillation period $P_c$ is recorded and the proportional, integral and derivative values adjusted accordingly using the table of Ziegler-Nichols method.

#### Charge control system

Bode plot of the system is in figure 9.8.

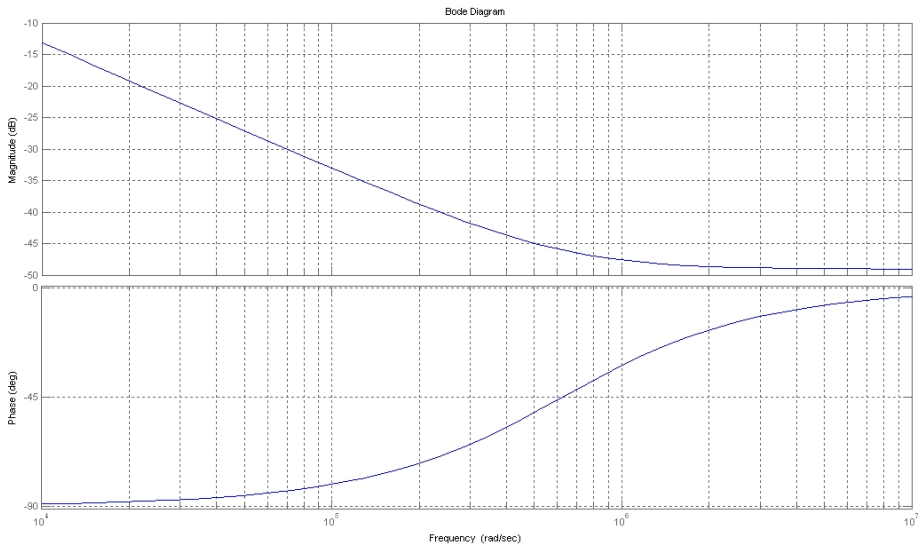



Figure 9.8. Bode diagram plot of the system

Closing the loop, Bode plot is in figure 9.9.
Starting from the dynamic specifications, the rising time of the controlled system

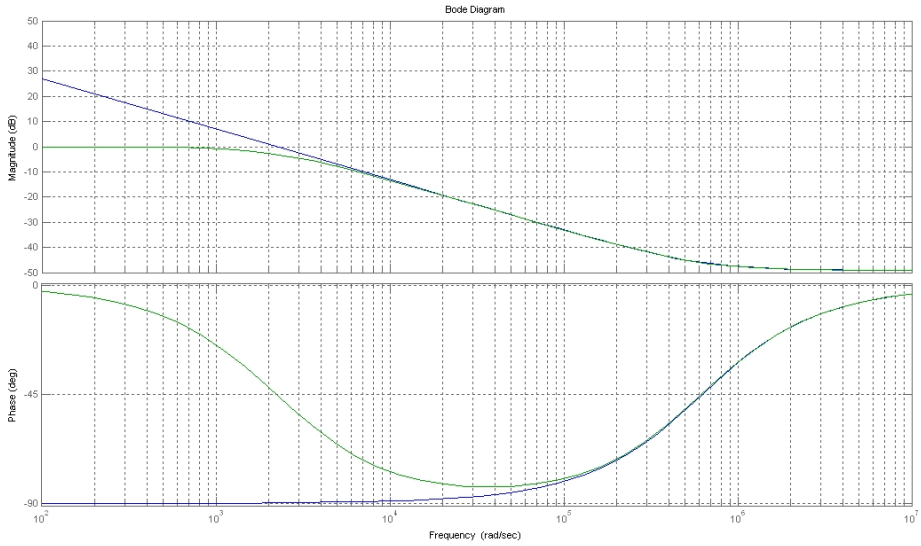



Figure 9.9. Bode diagram plot of the system

allows to determine the proportional term of the PI control:

$$\begin{cases} B = \dfrac{3}{T_{rise}} \\ \\ \omega_B = 2\pi B \end{cases} \Rightarrow K_p = bode\,(G_{cl}, \omega_B) \tag{9.7}$$

The integral term permits to guarantee null step response time error and further reduction of rise time, but can cause an overshoot output response. As previously mentioned, this term depends on proportional gain $K_p$ and sampling period $T_s$ considered.
Therefore, range of the two PI control terms are:

$$\begin{aligned} K_p &= 10 \div 30 \\ \\ K_i &= 0.001 \div 0.01 \end{aligned} \tag{9.8}$$

Using these values, the response time of the closed loop system is illustrated in figure 9.10.

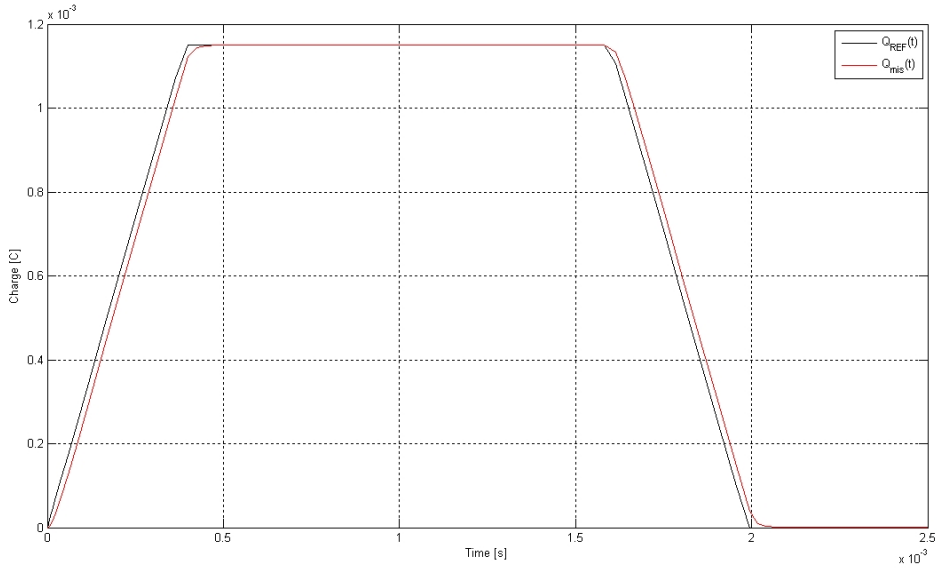



Figure 9.10. Simulated response time of the system

**Section Omitted**

For confidentiality reasons, due to patents in progress on the part of the private company that collaborates on this project, this section has been omitted.

## 9.3 Algorithms on Hardware

The developed system has been called `PZ0 Driver` (probably the next version will be called, originally, `PZ1 Driver`) and it is composed of an ECU, produced by ACTUA S.R.l, called `EKU 2.1` and a field module developed and produced at the mechatronic laboratory of the Politecnico di Torino.
As seen in PART I the `EKU 2.1` board is based on a fixed point Texas Instruments DSP (TMS320F2812) and an ALTERA FPGA (CycloneII EP2C35F672C8).

In a first stage of the project the current control loop was a simple peak current mode implemented partially on hardware:

- Sense resistor between the negative terminal of the load and the power ground
- Differential amplifier at the ends of the sense resistor
- Two comparators for peak and valley references
- Comparator on the piezoelectric voltage for the brake phase detection

and firmware on FPGA:

- Finite state machine for the duty cycle generation based on the outputs of the comparators
- Dead band time generation block
- Brake MOS management logic

The switching frequency of the current loop was $200KHz$.
The outer loop was a charge control loop, based on the PI compensator seen before, implemented on the DSP at a sampling frequency of $100KHz$.
This configuration had some limitations:

- Sub-harmonic instability
- High current ripple due to the low switching frequency
- Sampling period too high with respect to the bandwidth of the system

To solve the sub-harmonic instability some methods (available in literature), applicable with the hysteretic current control, can be implemented; in our project a digital method, omitted for confidentiality reasons, has been implemented.
Using that control on our system, it is possible to obtain a switching period of $3\mu S$; with the current control loop running at $333KHz$ it is also possible to increase the sampling period of the charge control loop but with our DSP it was not possible because even the simple PI regulator requires more time than the available one.
The only solution was to perform the `porting` of the outer loop from the DSP to the FPGA where we have not time problems. In this way we set the charge loop to a sampling period of $6\mu S \approx 166KHz$.
The FPGA implementation is completed with an IP-core performing the charge profile generation starting from couples of charge data sets and times to reach them, and two IP-cores for the management of the DAC and ADC devices.
This profile generation is a simple linear interpolation between the points $Q_i(kT)$

where $0 \leq i \leq 30$ is the data point number, $k$ is an integer and $T = 1\mu S$.
These parameters are sent by the DSP through the `DSP-FPGA Synchronous Memory Interface IP` of the FW-HRTOS used only in the direction from DSP to FPGA.
The only tasks of the DSP remain to communicate, by means of a CAN BUS channel, with a standard prototyping board in order to receive the profile parameters, and real time acquisition of the control variables in order to export them to a host PC by means of a tool called *RTCSuite* developed by ACTUA S.R.l. The sampling period for the acquisitions is $10\mu S$.
The CAN communication is performed during the DSP background while the acquisition is triggered from the host PC and performed in a dedicated service routine which takes up about $3\mu S$.
The time taken by the charge control loop (running on the FPGA) is about $100nS$; it is negligible with respect to the system time constants. The evaluation of the area taken up by an FPGA code is more important than the speed at which the code itself is running. Often in FPGA devices, the area occupancy and the maximal clock speed reachable are correlated because, in order to reach higher clock frequencies, it is necessary to add extra pipeline stages and it means more registers (thus more logic elements). For this reason some projects need a trade-off analysis between clock frequency and area occupancy that means used resources.
In this project the clock has a frequency of $100MHz$ (it is the main clock supplied by the FW-HRTOS) and the logic elements used are about 5500.

# Chapter 10

# Experimental Results and Model Validation

In this chapter is presented the experimental setup and, based on the results obtained on a test rig, a model validation is then performed.

## 10.1 Experimental Setup

The `PZO Driver` is constituted by a stack of different modules all of these, but the power module, compliant with the standard `pc/104`.
In picture 10.1 is showed this stack which is composed of (starting from the top):

- A pc/104 power supply module which generates the $+5V$, $+12V$ and $-12V$ starting from a voltage in the range $6V \div 40V$
- A communication module called `dCANi_AnyB 1.1` (by ACTUA S.R.l) which incorporates two isolated CAN BUS channels
- The `EKU 2.1` board (by ACTUA S.R.l) that is the logic kernel of the system
- Two power modules able to drive two piezoelectric transducers each

The EKU that has been used in developing this project is shown in pictures 2.2 and 2.3, in which the main components and parts are emphasized.

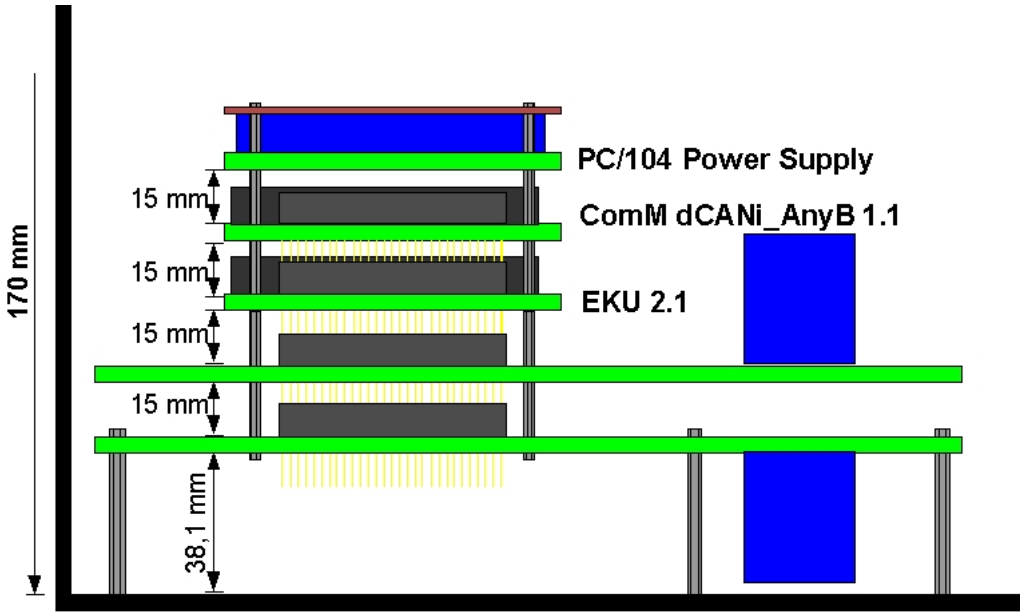



Figure 10.1. EKU stack used to prototype the control strategies

Part of the experimental tests have been performed using a standard electronic control unit which sent the charge profiles parameters by means of a dedicated CAN BUS channel.
CAN protocol cannot be considered a real time protocol but if it is used in a monotasking application and only in one direction some problems of non-determinism can be avoided.

The picture 10.2 shows the power module designed and developed in our laboratory. It implements the bidirectional buck converter (left side) able to drive two multiplexed loads by means of four selection MOS (up side).

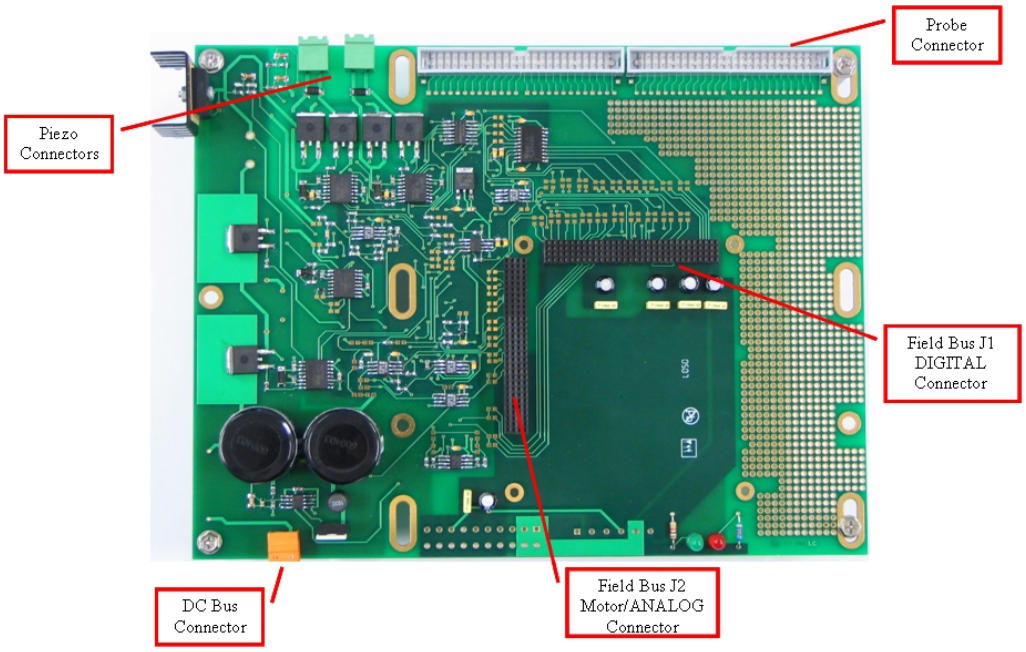



Figure 10.2. Power field module developed for the project

The `PZO Driver` stack has been mounted in a standard rack with the following external interfaces (see picture 10.3):

- Piezoelectric loads connectors. Loads have been mounted on a heavy metallic support in order to reduce the vibrations; on this support has been mounted a laser position sensor to acquire the tip displacement
- Power supply represented by an automotive battery. In the rack has been mounted a commercial boost to reach the DC-Link voltage of $200V$ required by the system starting from $12V$
- CAN connectors for standard communications
- Probe connectors to interface to a standard oscilloscope
- RS232 serial protocol connector to interface the system with a host PC in order to acquire all the internal variables by means of a tool called `RTC Suite` by ACTUA S.R.l

The goal is to characterize the kinematics of piezoelectric stack and identify the relation between charge and tip displacement of the piezoelectric transducer (see figure 10.4).

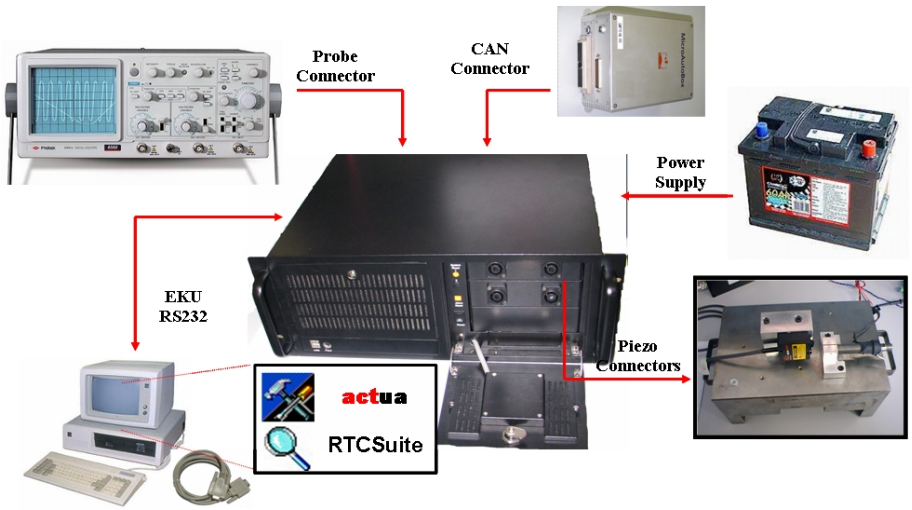



Figure 10.3. `PZ0 Driver` rack and its interfaces

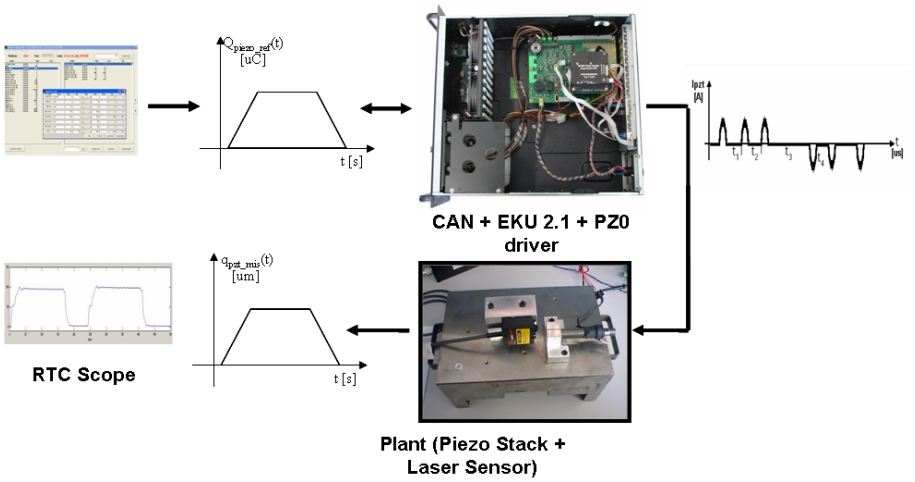



Figure 10.4. Test rig for piezoelectric stack characterization

`RTC Scope` is a tool included in the `RTC Suite`. It is a host-target solution for acquiring and visualizing data signals related to the real time variables available on `PZ0 Driver` platform.
It allows to acquire and visualize data signals on the host computer while real time code is running on target.
RTC Scope also allows personalizing scopes visualization mode just selecting between one of the available visualization types and adding/removing signals present

into each scope.
So with this tool it is possible to monitor all the real time variables present on the target (EKU 2.1) such as:

- DC Bus input voltage
- Voltage drop on piezoelectric actuator
- Current that flows in the piezoelectric element
- Charge stored on it
- Tip displacement obtained with the laser sensor
- All digital values and parameters in the control code

## 10.2 Model Validation

The aim of the project is to demonstrate that it is possible, given a piezoelectric stack, to determine the tip displacement value simply measuring charge and voltage on the piezoelectric element.
To validate the stack model it has been built a test bench where the external force acting on the stack is null (the stack tip is free to move without external load).

### 10.2.1 Piezoelectric Stack Dynamic Equations

As previously said 7.3 the dynamic equations of piezoelectric stack are:

$$\begin{cases} \underline{\underline{M}}\ddot{\underline{q}}_{stack}(t) + \underline{\underline{K}}^{Q}\underline{q}_{stack}(t) - \underline{\Gamma} Q_{piezo}(t) = \underline{F}_{ext}(t) \\ \underline{\Gamma}^{T}\underline{q}_{stack}(t) + \frac{1}{C_{pzt}} Q_{piezo}(t) = V_{pzt}(t) \end{cases} \tag{10.1}$$

Where $\underline{\underline{K}}^{Q}$ is the stiffness matrix obtained using charge driving.
Charge driving implies an open circuit in steady state conditions, so the output stage of power driver must have infinite impedance.

### 10.2.2 Tip Displacement Equations

The hypotheses to obtain an expression of tip displacement are:

- Steady state conditions: acceleration and velocity are considered approximately null, so it is possible to analyze the system without using the dynamics equations but the kinematics ones
- External force null

Under these hypotheses, considering only the first equation, the relation between charge and tip displacement is:

$$q_{pzt} = \underline{T} \left( \underline{\underline{K}}^{Q} \right)^{-1} \Gamma Q_{piezo} \tag{10.2}$$

It can be noticed that if the stiffness matrix and the coupling matrix are constant, there is a fixed relation between tip displacement and charge stored.
This term can be obtained experimentally doing a measurement of charge and tip displacement.
Once loaded the data it is possible to plot the results and obtain a relation between these two variables.

### 10.2.3 Static Characterization of piezoelectric stack

In the test bench the inputs are different charge profiles, while the outputs are the charge measured by means of the analog integrator and the tip displacement obtained using a laser sensor produced by `Micro Epsilon`.
Once acquired tip displacement at various charge values, it is possible to identify off-line the piezoelectric stack parameter that relate the two variables in a steady state condition with external force null.
Then, it is possible to fit a linear regression of plotted data points in order to find the best straight line through the data. In this case the slope is the piezoelectric stack constant that must be experimentally identified.
In the figure 10.5 the scheme of the experiments can be seen.

The procedure is:

- To set the control charge reference
- To measure the charge stored in the load at the end of the charging phase

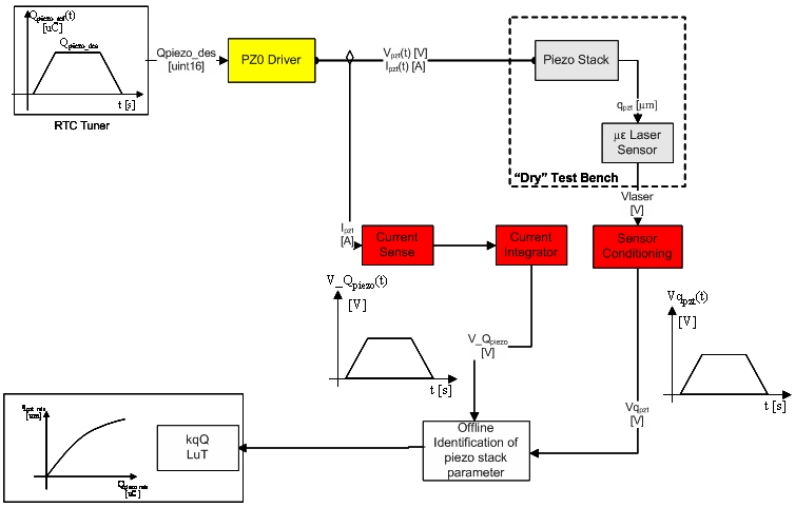



Figure 10.5. Schematization of the experiments

- To measure the voltage drop on piezoelectric stack at the end of the charging phase
- To measure the piezoelectric stack tip displacement at the end of the charging phase
- To estimate (model) the voltage drop on the load starting from the measured charge
- To estimate (model) the piezoelectric stack tip displacement starting from the measured charge
- To calculate the relative error between the estimated tip displacement and the measured one

The results obtained are shown in table 10.6 where a comparison has been done between the theoretical, or estimated, values and the experimental ones.

Then a linear regression of the ratio between the tip displacement and the stored charge has been performed and illustrated in figure 10.7.

Piezoelectric stack parameter remains stable with charge and displacement variations, so the idea of control tip displacement starting from charge loop could be a good solution.

| Measured | | | Estimated | | |
|---|---|---|---|---|---|
| Stored Charge [uC] | Piezo Voltage [V] | Tip Displacement [um] | Piezo Voltage [V] | Estimated Tip Position [um] | Position Error [%] |
| 250 | 65 | 22 | 66.22 | 21.12 | 4.01 |
| 350 | 76 | 29 | 76.24 | 27.51 | 5.15 |
| 450 | 86 | 34 | 86.26 | 33.90 | 0.30 |
| 550 | 96 | 40 | 96.28 | 40.29 | -0.72 |
| 650 | 106 | 47 | 106.30 | 46.68 | 0.69 |
| 750 | 120 | 54 | 116.32 | 53.07 | 1.73 |
| 850 | 126 | 59 | 126.34 | 59.46 | -0.78 |
| 950 | 134 | 64 | 136.36 | 65.85 | -2.89 |
| 1050 | 145 | 72 | 146.38 | 72.24 | -0.33 |
| 1150 | 157 | 80 | 156.40 | 78.63 | 1.72 |

Figure 10.6. Comparison between estimated and measured tip displacements related to charge values

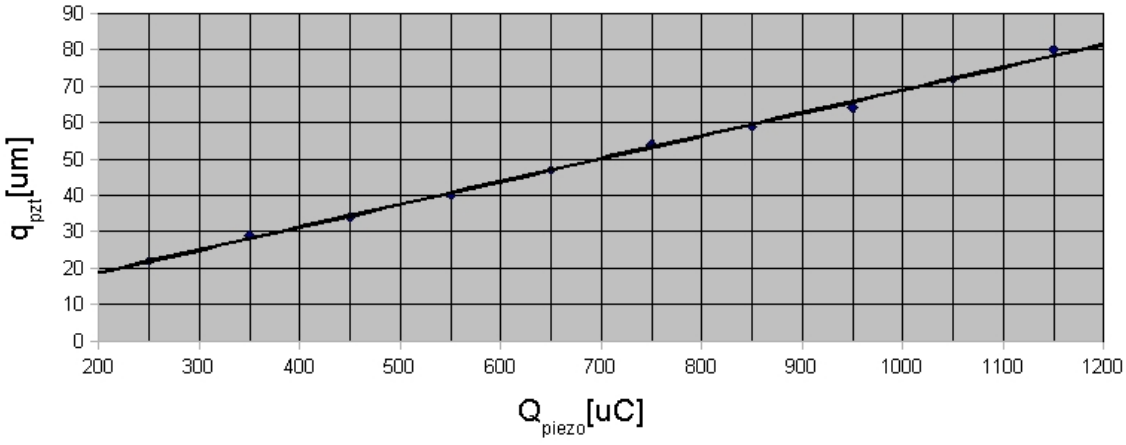



Figure 10.7. Linear regression of the ratio $q_{pzt}/Q_{piezo}$

### 10.2.4 Experiments on Hysteresis

It is possible to see the presence of the hysteresis in a piezoelectric stack plotting tip displacement or voltage drops referred to measured charge.
Test bench conditions are:

- Triangle charge profile
- Charge time $t_{ch} = 4000\mu S$
- Discharge time $t_{dis} = 4000\mu S$
- Charge set-point $Q_{piezo\_ref} = 1150\mu C$

In the diagram 10.8 are plotted:

- Charge reference $Q_{ref}(t)$ and charge measure $Q_{mis}(t)$
- Charge stored $Q_{piezo}(t)$ referred to the Tip displacement $q_{pzt}(t)$
- Voltage drop on load $V_{pzt}(t)$ referred to Tip displacement $q_{pzt}(t)$

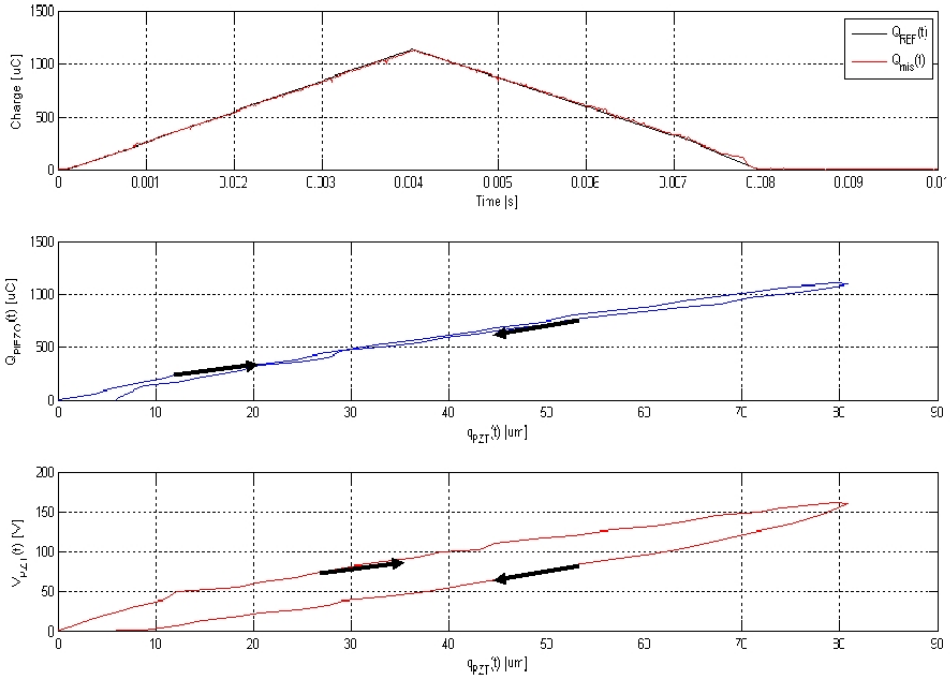



Figure 10.8. Hysteresis evaluation from acquired data

The second plot highlights what seen in 6.1.2: driving piezoelectric elements with a charge feedback the hysteresis brings to a little value (about the resolution of the position sensor); so introducing the charge loop it is possible to control the piezoelectric tip displacement starting from the charge measured. In the third plot the hysteresis between the voltage drop on the load and the tip displacement is evident.
This hysteresis phenomenon is also visible performing step modulated profiles: in the figure 10.9 it is possible to notice how the tip displacement follows the charge trend while the voltage drop has different values at the same conditions of charge and displacement of the stack tip.

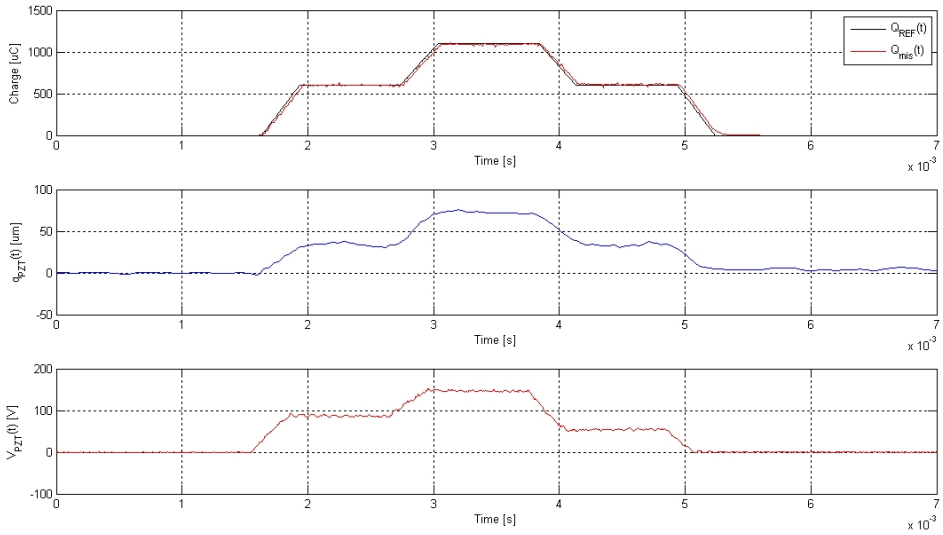



Figure 10.9. Step modulated charge profile

# Bibliography


[1] Soldano M. Brown R. One cycle control ic simplifies the pfc designs. *IEEE APEC*.

[2] Frank Vahid Dean Tullsen Roman Lysecky. David Sheldon, Rakesh Kumar. Conjoining soft-core fpga processors.

[3] dSPACE. Microautobox on-line documentation. *http://www.dspaceinc.com/ww/en/inc/home/products/hw/micautob.cfm?nv=n2*.

[4] dSPACE. Rapidpro hardware on-line documentation. *http://www.dspaceinc.com/ww/en/inc/home/products/hw/rapidpro.cfm?nv=n2*.

[5] A. Tonoli. E. Brusa, S. Carabelli. Self-selsing colocated structures with distribuite piezoelectric transducers. *Proc. Of 7th Int. Conf. on Adaptive Structures and Technology, Rome*.

[6] Maddaleno F. Alimentatori a commutazione v. ii. *Politeko*.

[7] Richard Fryer. Fpga based cpu instrumentation for hard real-time embedded system testing.

[8] Comstock R H. Charge control of piezoelectric actuators to reduce hysteresis effects. *US Patent Specification 4263527*.

[9] Robert W. Brodersen University of California at Berkeley. Hayden Kwok-Hay So. Borph: An operating system for fpga-based reconfigurable computers.

[10] IEEE inc. Ieee standard on piezoelectricity. *ANSI/IEEE Standard 176-1987*.

[11] National Instruments. Compactrio on-line documentation. *http://www.ni.com/pdf/products/us/cat_crio907x.pdf*.

[12] Karlsruhe and Palmbach Physik Instrumente (PI). Designing with piezoelectric transducers: Nanopositioning fundamentals.

[13] Dixon L. Average current mode control of switching power supplies. *TOPIC5*.

[14] OMalley K. Levin G. Designing with hysteretic current-mode control. *EDN*.

[15] Cruz-Hernandez J M and Hayward V. An approach to reduction of hysteresis in smart materials. *Proc. IEEE Int. Conf. Robot. Automat. pp 1510-5*.

[16] D. Shand Nallatech. M. Devlin. Scaling fpga systems for software radio.

[17] J. Gregory Steffan. Peter Yiannacouras, Jonathan Rose. The microarchitecture of fpgabased soft processors.

[18] Physik Instrumente (PI). Web page. *http://www.physikinstrumente.com*.

[19] Sokal N. Redl R. Current-mode control, five different types used with the three basic classes of power converters. *PESC 85, p. 771-785.*
[20] Thorlabs. Web page. *http://www.thorlabs.com.*
[21] James O. Hamblen. Tyson S. Hall. Using an fpga processor core and embedded linux for senior design projects.
[22] B J G Vautier and S O R Moheimani. Charge driven piezoelectric actuators for structural vibration control: issues and implementation.
[23] Ralph D. Wittig. Onechip: An fpga processor with reconfigurable logic.